

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

(2)

AD-A278 943



To average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering the collection of information, Send comments regarding this burden estimate or any other aspect of this form to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Avenue, Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

DATE

3. REPORT TYPE AND DATES COVERED

FINAL/01 OCT 89 TO 30 JUN 93

4. TITLE AND SUBTITLE

CASE BASED REASONING IN ENGINEERING DESIGN (U)

5. FUNDING NUMBERS

6. AUTHOR(S)

Professor K Sycara

6895/DARPA
F49620-90-C-0003

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The Robotics Institute
Carnegie-Mellon University
Pittsburgh, PA 15213-38908. PERFORMING ORGANIZATION
REPORT NUMBER

AFOSR-JR- 94 0280

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFOSR/NM
110 DUNCAN AVE, SUITE B115
BOLLING AFB DC 20332-000110. SPONSORING / MONITORING
AGENCY REPORT NUMBER

F49620-90-C-0003

94-13604



13410

11. SUPPLEMENTARY NOTES

DTIC
ELECTE
MAY 06 1994
S G D

12a. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED

12b. DISTRIBUTION CODE

UL

13. ABSTRACT (Maximum 200 words)

Case-Based Problem Solving is based on the idea that problem solving should re-use solutions and other information from previously solved problems instead of relying solely on a base of procedures or rules. The researchers presented a case-based design system, CADET retrieves and re-uses previous successful designs while avoiding previous failures such as poor materials or high cost. The system uses certain behavior-preserving transformation techniques to transform an abstract description of the desired behavior of the device into a description that can be used to find relevant designs in memory. This approach, in effect, decomposes given behavior specifications into "sub-behaviors", making it possible to recognize parts of previous designs that can be synthesized to form a new device. Currently, the system can perform conceptual design of mechanical devices that exhibit continuous and reciprocating behavior. In addition, since CADET can generate a wide variety of behaviorally equivalent alternative designs for a given set of design specifications, it can be used as a designer's brainstorming assistant.

14. SUBJECT TERMS

15. NUMBER OF PAGES

64

16. PRICE CODE

17. SECURITY CLASSIFICATION
OF REPORT
UNCLASSIFIED18. SECURITY CLASSIFICATION
OF THIS PAGE
UNCLASSIFIED19. SECURITY CLASSIFICATION
OF ABSTRACT
UNCLASSIFIED20. LIMITATION OF ABSTRACT
SAR(SAME AS REPORT)

94 5 05 090

FINAL REPORTApproved for public release;
distribution unlimited.**Case Based Reasoning in Engineering Design**ARPA Order 6895
Program Code 9E20Contractor: Carnegie Mellon University
Program Manager: Abraham Waksman (202) 767-5025

Principal Investigators: K. Sycara (412) 268-8825, D. Navin-Chandra 268-7019

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. 6895

Monitored by AFOSR Under Contract No. F49620-90-C-0003

Accession For	
NTIS CRA&I	☒
DTIC TAB	☒
Unannounced	☐
Justification _____	
By _____	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

Table of Contents

1. Introduction	2
1.1 Design	5
1.2 Overview of Case Based Design	7
1.3 Relationship to other work	9
2. Case Based Reasoning in Engineering Design	12
2.4 Representing Behavior in Design Cases	12
2.4.1 Behavior and Influences	12
2.4.2 Total Influences	15
2.5 Index Transformation	16
2.5.1 Two Rules for Reasoning about Influences	17
2.6 Using the design rules	19
2.6.1 Using Domain Laws	19
2.6.2 Hypothesizing new variables	20
2.7 Case Matching	23
2.8 Constraint Checking	24
2.9 Concluding Remarks	25
APPENDIX A. Conceptual Design Example	26
2.9.1 The Design Process	27
APPENDIX B. Proof of the Analysis Theorems	32
3. The CADET system	34
3.10 CADET System Architecture	35
3.11 CADET's Transformational Approach	37
3.12 Matching and Retrieval of Cases and Snippets	41
3.13 Material Adaptation	44
3.13.1 Case Representation	45
3.13.2 Material Adaptation Process	47
3.14 Conclusion	50
3.14.1 A Metal Bending Device	59
4. Reasoning about Connections in Synthesis	64
4.15 Introduction	64
4.16 A Case-Based Design Tool	66
4.17 Physical Synthesis of Design Cases	69
4.18 Selection of Connections	73
4.18.1 Kinematic Inputs	74
4.18.2 Connectivity Conditions	76
4.19 Example	78
4.20 Conclusions	80
5. Representation and Reasoning about Performance	85
5.21 Introduction	85
5.22 Behavior Representation in CADET	86
5.23 Second-Order Behavior Influence Representation	89
5.24 Representation and Reasoning about Performance	90
5.24.1 Ranges of Operation	91
5.24.2 Type of Variation of Behavior Variables	93
6. CARD: The CADET casebase	97
6.25 Introduction	97
6.26 Description of CADET	98

6.26.1 Case representation	99
6.26.2 Case Retrieval	106
6.27 Getting Started	106
6.27.1 Sources	106
6.27.2 Setting up the database	106
6.27.3 Setting up the system	107
6.27.4 Using the system	107
6.28 Bugs present and Fixes	109
6.29 Index transformation and Retrieval	109
6.29.1 Elaboration tree	112
6.29.2 Retrieval tree	114
6.30 Useful functions	114
7. The Lisp-C-ESQL interface to a Informix Relational Database & CADET Case input and storage mechanisms	116
7.31 Introduction	116
7.31.1 Implementation issues	117
7.32 Description of LISP-SQL functions	117
7.32.1 Error messages	118
7.32.2 Usage of the functions	119
7.33 Cadet case input	119
7.34 An example session	119
Bibliography	122

People Involved in the Project over the years:

S. Narasimhan, CMU, PhD Candidate

T. Madhusudan, CMU, PhD Candidate

J-L Koning, INRIA, France (Visiting Scientist)

N. Michelena, U. of Michigan (was Post-Doc at CMU)

K. Sycara, CMU, SCS Faculty

D. Navin-Chandra, CMU, SCS Faculty

R. Guttal, RPI, (M.S. from CMU)

Chapter 1

Introduction

Case-Based Problem Solving is based on the idea that problem solving should re-use solutions and other information from previously solved problems instead of relying solely on a base of procedures or rules. We present a case-based design system, CADET, that functions as a designer's assistant for mechanical design. CADET retrieves and re-uses previous successful designs while avoiding previous failures such as poor materials or high cost. The system uses certain behavior-preserving transformation techniques to transform an abstract description of the desired behavior of the device into a description that can be used to find relevant designs in memory. This approach, in effect, decomposes given behavior specifications into "sub-behaviors", making it possible to recognize parts of previous designs that can be synthesized to form a new device. Currently, the system can perform conceptual design of mechanical devices that exhibit continuous and reciprocating behavior. In addition, since CADET can generate a wide variety of behaviorally equivalent alternative designs for a given set of design specifications, it can be used as a designer's brainstorming assistant.

Case-Based Reasoning (CBR) is the problem solving paradigm where previous experiences are used to guide problem solving [5, 29, 57, 61, 17]. Cases similar to the current problem are retrieved from memory, the best case is selected from those retrieved and compared to the current problem. The precedent case is adapted to fit the current situation based on the identified differences between the precedent and the current case. Successful cases are stored so they can be retrieved and re-used in the future. Failed cases are also stored so that they will warn the problem solver of potential difficulties and help recover from failures. If a current case has features similar to a past failure, then the problem solver is warned not to attempt the failed solution. After the problem is solved, the case memory is updated with the new experience. Thus, learning is integrated with problem solving.

We have identified some characteristics of domains where CBR is applicable.

1. An expert knows what he/she means by a case.
2. Domain experts draw inferences from comparing a current problem to cases.
3. Experts adapt cases to solve new problems.
4. Cases are available in bibliographic sources, in experts' memories, or can be recorded as new solutions are attempted.

5. There are means in the domain to assign an outcome to a case, explain it and deem it a success or a failure.
6. Cases can be generalized to some extent. Features that make them relevant can be abstracted.
7. Comparisons can be implemented computationally with some level of effectiveness.
8. Cases retain currency for relatively long time intervals.
9. The domain may, or may not have a strong model.
10. Cases are used in training professionals in the domain.

Engineering design meets all the above criteria. In particular:

1. A case is a previous design of an artifact.
2. Design experts generate designs largely from prior cases, and use analytical models to verify that the generated design meets its specifications.
3. Design specifications, simulation, and prototyping results guide adaptation of design cases.
4. Design cases are readily available in design catalogs and record books. The catalogs provide information about a wide variety of devices, their parts, characteristics, materials, uses and behavior. Companies that keep records of the designs they generate, try to re-use the designs when similar tasks or problems are encountered.
5. Simulations, prototypes, and field tests are means by which designs are tested and evaluated.
6. Dissimilar structural configurations can deliver the same behavior. Hence, behavioral descriptions are natural abstractions in design. (Section 4 presents our approach that is based on behavioral abstractions).
7. Case comparisons and adaptations can be done effectively (Sections 5, and 6 present a system that addresses these issues).
8. Designs retain currency for long periods of time. For example, the basic design of a toaster has not changed for 10 years. Technological innovations may cause design adaptations.
9. Despite the existence of physical laws and principles, design is a creative and poorly understood process.
10. Engineering students are taught design through the use of numerous cases. When an entry-level engineer joins a design office, an important part of his training involves going through the design records of previous projects.

Although engineering design is a domain amenable to case-based techniques, many challenging issues must be addressed. First, is the issue of relating behavior and structure. During the design process, a designer transforms an abstract functional description for a device into a physical description that satisfies the functional requirements. In this sense, design is a transformation

from the functional domain to the physical domain. In order to effect this transformation, a designer needs to reason at different levels of abstraction ranging from the physical to the functional. For example, while trying to produce a design to perform a particular function, functional descriptions may be used to retrieve cases. However, using a case to physically synthesize the design, involves extracting appropriate physical features from the case. To support such reasoning, it is necessary to develop design case representations in terms of vocabularies that express and capture relationships between device function, behavior and structure [65]. In addition, we need inference strategies that can utilize these representations and integrate the results. Second, good mechanical designs are often highly integrated, tightly coupled collections of interacting components with no obvious decomposition of the overall function into subfunctions. Previous cases can represent good solutions to these interactions and can be profitably re-used. The major challenge for this issue is developing indexing schemes in terms of tightly coupled design features. Third, the initial functional description of the artifact is usually underspecified so that a designer must have means to identify information "gaps" during the design process and generate new problem solving subgoals to resolve them. In a case-based framework, these dynamically generated subgoals can give rise to indices for retrieving cases to fill the gaps. This is a common problem in design and was encountered in the development of one of the first applications of CBR to engineering design [43]. Fourth, a complete design is synthesized from solutions to subproblems that capture desired subfunctions of the artifact. These subproblem solutions can be expressed as pieces (snippets¹) of previous designs that must be independently accessible. In addition, the behavior exhibited by the combination (synthesis) of the retrieved snippets must be equivalent to the desired overall device behavior. Considerable complication arises from the fact that although a design might be verified to be correct at the behavioral level, simulation at the physical level might fail. The problem solver must synthesize device pieces at one level of abstraction while making sure the parts will work together in physically correct ways. In addition, verifying that each component part meets its specifications does not guarantee that the design as a whole will meet its specifications. Thus, both partial and complete designs must be verified.

We have developed a system, called CADET, that performs conceptual design by synthesizing a device from snippets accessed from previous design cases. CADET uses a multi-layered representation to express function, behavior, structure and related constraints. Case representations are distributed between an object oriented system and a commercial relational database. We have developed a fast algorithm that allows us to perform case matching using embedded-SQL commands. The case base is currently populated with about 75 cases. About a fourth of these cases are taken from the book *The way things work*. The rest are taken from commercial catalogs recommended to us by designers. These include *Level and Flow Sensors* (Flowmem Industries) and the *Fluidpower* catalogs and handbooks from Parker Industries.

¹As coined by Janet Kolodner

1.1 Design

Design is the process of generating a description of an artifact that satisfies given specifications (goals and constraints) which describe the desired function of the artifact. Design can be viewed as a transformation from the functional domain to the physical domain [40, 58, 52]. In order to effect such transformations, a problem solver must be able to reason at different levels of abstraction from the functional to the physical. Such reasoning remains opaque in the domain of mechanical systems design since the transformational process is neither well-characterized nor understood. This is partly due to the fact that, in contrast to other design domains such as software engineering and circuit design [42], good mechanical designs are often highly integrated, tightly coupled collections of interacting components [60, 74]. A simple and obvious correspondence between specific functional requirements of the artifact and individual components in the design does not usually exist. For example, in a can opener, the circular blades perform the function of holding the can, rotating it and cutting off the top. Ironically, these are also the major functions of the entire can opener. It is not possible to identify specific features of the can opener or its blades which perform each of the functions independently.

Due to the tightly coupled and interacting nature of mechanical designs, reasoning from prior design cases is proving to be a suitable design methodology as opposed to direct "decompose and recombine" strategies that have successfully been utilized in VLSI design [58, 72]. Case-based problem solving is based on the premise that a machine problem solver make use of its experiences (cases) in solving new problems instead of solving every new problem from scratch [29]. Design cases reflect good design principles, such as function sharing [60] and incorporate decisions that take advantage of, or compensate for incidental components interactions. Our investigation has been conducted within a framework of a case based reasoning methodology for mechanical design and has been implemented in the CADET system (Case-based Design Tool). CADET operates in the domain of hydro-mechanical devices such as faucets, flush tanks, valves, and pumps. It has a memory of previous designs and components² that guide the design process in producing new designs. Cases are represented using a multi-layered representation which includes structural features of the artifact, functional features and relations, and linguistic descriptors. Cases can be retrieved from memory using a variety of indices corresponding to the above mentioned features.

The case memory stores both devices and device components as cases. The cases have associated descriptions both in terms of behavioral³ and structural characteristics. If parts of the

²We use the word "component" to refer to cases, or pieces of larger cases that may perform subfunctions of a particular device but are not necessarily standard components.

³In this paper, we concentrate on the physical behavior of devices. Though we use the words function and behavior interchangeably; the function of a device strictly refers to the way it interfaces with the outside world. For example, a watch has the behavior of moving it's hour and minute hands, while it has the function of telling time. Many devices can have the same function but may have different behaviors. For example, digital watches and sun-dials also tell time.

behavioral specification of the desired design correspond to the behavioral indices of the components, then the components (and hence their structural descriptions) can be directly accessed. However, because there is no one-to-one correspondence between the desired behavior of a device and the individual component behaviors, it is often not possible to find relevant cases by using the given overall behavioral specification as an index into case memory. This gives rise to the need for techniques to transform an abstract description of the desired behavior of the device into a description that provides indices with which relevant components can be retrieved. One way to do this in the domain of mechanical systems is to use transformations that convert the given specifications into alternative forms that facilitate the identification of relevant components and the subsequent realization of the design.

Behavior preserving transformations require formal representations of the behavioral specifications of mechanical systems as well as formal representations of behavioral characteristics of mechanical components. The representation that we use is the language of qualitative physics [33]. Qualitative physics has predominantly been used in analyzing the behavior of existing systems. We, however, use these techniques for the design of new systems by transforming given behavior specifications. The transformational approach decomposes given behavior specifications into "sub-behaviors" whose composition preserves the overall desired behavior. The decompositions do not impose any a priori structure or topology on the physical realization of the design. The decompositions are collections of sub-behaviors with information on how the sub-behaviors must interact to produce the overall device behavior. In effect, we do not require an a priori decomposition of design specifications.

To accomplish the task of extracting relevant portions of a design, the problem solver must be able to recognize that the flush tank's behavior can be achieved by combining relevant "sub-behaviors" some of which are also present in the faucet. In other words, the relevant capability of the problem solver is to recognize shared "sub-behaviors" among devices which are not given a priori.

We present an approach to accomplish recognition of shared "sub-behaviors" based on behavior preserving transformations that uses and extends qualitative reasoning about physical systems.

The rest of the paper is organized as follows: the next section provides an overview of our approach to case-based design. Next, we present a graph-based representation of behavior and discuss some properties of the representation. This is followed by a presentation of the notion of behavior preserving transformations, and a discussion of how these transformations serve as design rules that help generate synthesis strategies. Finally, we will examine the underlying case matching and behavioral-constraint checking methods. An extended conceptual design example outlining the problem solving steps taken by CADET is presented in the first appendix. The second appendix provides assumptions and proofs underlying the transformational approach used in CADET.

1.2 Overview of Case Based Design

Designing is pervasive in many human activities, for example an engineer conceiving of a new type of toaster, two parties negotiating a contract, a financial manager configuring a profitable portfolio, or a chef concocting a new dish. Underlying these design tasks is a core set of principles, rules, laws and techniques which the designer uses for problem solving. The designer's expertise lies in his ability to use these techniques to produce a feasible design. The designer's expertise is a consequence of his experience and training, much of which is based on previous exposure to similar design problems. This is particularly true in our domain of interest: engineering design [48, 16, 19].

How important are cases in Engineering Design? Cases are the primary way in which engineering students are taught to design. This is because there are no general algorithms for design. Typically, students are exposed to numerous cases and examples which illustrate how complex problems are solved. Even when an entry-level engineer joins a design office, an important part of his training involves going through the design records of previous projects. Although the engineering design community recognizes the importance of cases in problem solving, the use of precedent cases in Computer-Aided Engineering (CAE) tools has been largely ignored. This is not because the CAE research community is not aware of the ubiquity of case based reasoning in design, but because they have not had access to the right techniques.

A typical CAE tool for design includes a geometric modeling system and a standard set of analytic tools for tasks such as finite-element and boundary-element analysis. Over the last five years, design tools have been extended to include some design heuristics. These heuristics come in several forms: as rules, as constraints and as recommendations. It is only very recently that a third aspect of the design process, the use of past cases, is beginning to be recognized in the Design Automation literature [41, 11, 73, 23, 42, 35, 45, 22]. Cases provide memories of past solutions that have been used successfully. They also provide memories of past failures and repairs which can be used to warn the problem solver of impending problems and to repair failures without having to work from scratch [17, 28, 62, 63].

Another advantage of case based design is in the area of knowledge acquisition. We have found that asking a designer to give us examples of cases he has worked on, is much easier than asking him to give us a list of rules that he uses to design. There is experimental evidence that designers don't explicitly think in terms of rules [56]. In addition, design case studies are also readily available in the literature. Based on interviews with designers we have identified certain commercial catalogs of hydromechanical devices. These catalogs provide information about a wide variety of devices, their parts, characteristics, materials, uses and behavior. We are using these catalogs as sources of cases. The case base is populated with about 75 cases. About a fourth of these cases are taken from the book *The way things work*. The rest are taken from commercial catalogs recommended to us by designers. These include *Level and Flow Sensors* (Flowmem Industries) and the *Fluidpower* catalogs and handbooks from Parker Industries.

There are two broad groups of issues that must be addressed in any case-based reasoning system:

(1) how to represent and index cases in the case memory, and (2) how to use cases in problem solving. Previous research in index determination in the CBR literature [26, 18] has identified goals as well as object attributes as general classes of features that can be used as indices. Since artifacts always have intended functions, the functional specification gives rise to a set of related indices.

In dealing with physical domains, a case based problem solver must be able to work at several levels of abstraction ranging from the physical to the functional level. For example, while trying to produce a design to perform a particular function, functional descriptions may be used to retrieve cases. However, to use a case to physically synthesize the design involves extracting appropriate physical features from the case. This requires that the representation be able to capture the relationship between physical form and qualitative function. CADET uses a multi-layered representation that gives rise to related indices. The representational abstractions that are used for indexing the case memory are: (1) Linguistic descriptions, (2) Functional Block Diagramming, (3) Qualitative influence graphs, and (4) Qualitative States [66]. This case representation is distributed between an object oriented system and a commercial relational database manager. We have developed a fast algorithm that allows us to perform case matching using embedded-SQL commands.

The CADET system has a synthesis module, an evaluation module and a knowledge base organized around a design black-board which is managed by a Control Module. The Control Module has access to three types of synthesis methods: rule-based, case-based and search-based. The case synthesis sub-module can accept a design from the Design Black-Board and add new components drawn from cases in the Case Knowledge Base. Synthesized alternatives are evaluated in the Evaluation Module through the use of cases, qualitative simulation, and constraint checking. If bugs or "gaps" are found in the design alternatives, they are either considered for debugging or are discarded. The process of case based design consists of the following steps that are iteratively applied as new subgoals are generated during problem solving [65, 67]. Though we present these steps sequentially, they are interleaved during problem solving.

1. *Development of a Functional Description.* At the simplest level, the desired artifact can be viewed as a black-box which takes certain inputs and produces desired outputs. The function of the black-box is described by qualitative relations explaining how the inputs and outputs are related. The system's job is to realize an artifact which will convert the inputs into the desired outputs.
2. *Retrieval of Cases.* A set of design cases (or case parts) bearing similarity to a given collection of features are accessed and retrieved. Retrieval is performed using not only the existing features of the input specification, but also indices arising from index transformation strategies (the focus of this paper).
3. *Development of a Synthesis Strategy.* A synthesis strategy is a description of how the various cases and case pieces will fit together to yield a working design.
4. *Physical Synthesis.* Realization of the synthesis strategy at a physical level. This is a difficult problem since undesirable interactions among case parts may occur.

In addition, since it is very rare to retrieve cases that exactly match the design specifications, cases and case pieces must be physically modified before actual synthesis.

5. *Verification.* Adverse interactions could lead to non-conformance of the design to the desired specifications. This is verified through quantitative and qualitative simulation. If the simulation is correct, and if all the constraints are satisfied, then the design is successful. If not, repair (next step) is attempted.
6. *Debugging.* Debugging involves a process of asking relevant questions and modifying them based on a causal explanation of the bug. These questions serve as cues into memory.

1.3 Relationship to other work

A variety of CBR techniques for case representation and reasoning in design have been suggested in the literature: (1) Causal representation of prior design problems and solutions in the architecture domain was used in the CYCLOPS system [43]. Such causal networks have also been used in the medical domain [32]. (2) In the meal planning/design domain, JULIA uses plans and sub-plans represented as frames with slots for the different courses of the meal [20]. In the engineering design domain, deep models have been used successfully [55]. We will discuss and compare five systems: (1) Architectural design systems STRUPLE and ARCHIE. (2) Motion synthesis by connecting the inputs and outputs of primitive mechanisms. (3) *Ibis*, a system that connects a given set of primitives to satisfy a given goal⁴ (4) The behavioral component-substance based modeling and reasoning in KRITIK.

In STRUPLE, experience is stored in the form of descriptions of building design solutions [35]. Matching is done using a similarity metric that compares significant common aspects of the matched buildings and the current building. A matching criteria is a requirement of similarity imposed on a feature of a matching building. For example, the number of stories, the intended use, the design wind load etc. Each matching building is ranked to measure how well it resembles the current building according to both required and desired criteria. The method is similar to measuring the relative error of two function values. STRUPLE's similarity metric is based on a fixed set of criteria that does not consider the rationale involved in the decision process. Since it involves only specific domain features, STRUPLE's index mechanism is unable to find analogies across domains. The ARCHIE system is an architectural design system for office buildings [14]. It also uses a flat, frame based representation of cases (there is no deep reasoning about shape and form). ARCHIE's contribution lies in its use of qualitative domain models for retrieving cases. For example, it has a model of how various features of an office space (e.g. wall color, lighting quality) affect the lighting quality of the built environment. Such a model can be used to evaluate a design concept and to retrieve all prior cases where a similar problem was encountered. This idea is similar to how causal models were used in the

⁴This goes beyond motion synthesis by including functional parameters such as pressures and flow rates.

CYCLOPS system that retrieved cases to debug landscape architectural layouts. For example, a noise problem may be fixed by using trees or acoustic barriers taken from a prior case. CYCLOPS, however, used ad-hoc models for each case. ARCHIE, on the other hand, is able to work with several, central domain models. This makes the approach more general.

Mechanical designs can be viewed as being synthesized from conceptual building blocks that perform specific kinematic functions [31]. The motion synthesis approach provides a method for recognizing a given behavior in terms of known primitive behaviors. This is one of the first formalized ways of viewing design as the synthesis of kinematic processes, however, the approach is limited to a fixed set of primitives. CADET, on the other hand, is able to reach into a large casebase and select pieces of cases dynamically. The notion of synthesizing devices from known components is extended beyond basic kinematics in the Ibis system [78, 79]. In Ibis, components are represented as sets of interactions among behavioral parameters of the component. This approach allows one to use any aspect of a behavior, not restricting behavior descriptions to just one domain (e.g. qualitative motion synthesis). Ibis' major drawback is that its problem solving ability depends on the syntactic form of the goal. Because the program suffers from a Functional Fixedness [36, 8], it cannot recognize behavioral equivalence between a given index and a case if they are not syntactically similar. CADET's transformation based approach, on the other hand, adequately addresses this problem. If CADET cannot find a direct equivalence, it looks for behavioral similarities. Through the process of influence hypothesis and matching, the system is able to use physical laws and principles embedded in prior design cases to achieve its current goals. In this way, CADET is opportunistic about the principles it exploits in a design. This is unlike other approaches which assume that all the relevant principles have been identified *a priori*, as in the Ibis system. Because CADET hypothesizes influences, it does not limit itself to the given knowledge. Consequently, it can generate elaborations that represent designs that have never been conceived of before. Further, CADET's ability to recognize behavioral equivalences reduce its sensitivity to the form of the problem description.

KRITIK is another deep model based design system [12]. It uses a component-substance model that captures the components (e.g. battery, pipe), substances (e.g. water, electricity) and relations (e.g. containment, connection). Behavior of such systems are represented as graphs of states and transitions. When the system is given a design task, it retrieves the best case and deduces modifications that can be made. Modifications involve changes in relations, substitution of substances, parametric changes of components etc. The CADET approach is quite different. We use a representation that has no structure in the behavior description. This allows us to transform behavior descriptions in a principled way. We believe that the space of elaboration of behaviors is complete (proving this is a whole dissertation in itself.) Another advantage of not committing to structure, is CADET's ability to mix, match and re-use whole and parts of many prior cases to solve a given problem. The use of multiple cases has been shown to be important in advanced CBR systems [50].

Currently CADET can perform conceptual design synthesis of continuous and reciprocating devices. It cannot synthesize a new feedback design from components that do not have any feedback. If however, CADET's casebase were to contain feedback devices, it will be able to retrieve and use them. We are currently extending the transformational synthesis approach to feedback devices.

Chapter 2

Case Based Reasoning in Engineering Design

Design is not done in a vacuum. Engineers often rely on prior designs to make new design decisions instead of solving every new problem from scratch. Prior designs that represent good solutions to the tightly coupled nature of mechanical devices are used as guides. Moreover, prior failures are used to avoid repeating old mistakes. In this paper we present a computer based approach to exploiting the knowledge embodied in prior designs. Reasoning from design cases requires the ability to use cases, or pieces of cases that realize subfunctions of the device being designed. It is, however, difficult to recognize and retrieve relevant cases or case pieces using a given design specification. Because there is no one-to-one correspondence between the desired behavior of a device and the individual component behaviors, it is often not possible to find relevant design cases by using the given overall behavioral specification as an index into case memory. We approach this problem by elaborating the given behavior specification into a description that gives rise to indices with which relevant components can be retrieved. The elaborations are carried out in a *behavior-preserving* manner using two transformation operators that (a) rely on physical laws if it is known which ones are relevant, or (b) hypothesize behaviors and then search the case memory for ways in which the required behaviors may be achieved. These two approaches are used opportunistically in CADET, a case-based mechanical design system.

2.4 Representing Behavior in Design Cases

In dealing with physical systems a reasoner needs to retrieve cases based not only on the *physical attributes* of a device but also on its *functional behavior* [13]. CADET represents designs at several levels of abstraction ranging from the physical to the functional level [66]. Behavior is represented in terms of qualitative influences.

2.4.1 Behavior and Influences

In CADET, device behavior is represented as a collection of influences among the various inputs

and outputs⁵. An influence is a qualitative differential (partial or total) *relation* between two variables one of which is a dependent variable and the other an independent variable. The notion of influences is based on the notion of confluences [6] and causality [24] in device behavior.

Influences are organized as graphs. In general, an influence graph is a directed graph whose nodes represent the variables of concern and whose arcs are qualitative values depicting the influence of one variable on another. These graphs of influences are used to represent the behavior of devices, where each influence corresponds to some physical law or principle.

Consider, for example, a household water tap that has two inputs: a water source and a signal to regulate the rate of flow of water. In the two and a half dimensional representation [74] that we have adopted, the tap is represented as a pipe with a gate as shown in Figure 2-1. The flow rate is given by Q and the position of the gate is given by X . The position of the gate controls the flow rate. This behavior is represented as an influence $Q \leftarrow^+ X$, which is read as follows: "The flow rate (Q) increases (+) monotonically with an increase in the signal (X)". This influence represents the "tap" principle.

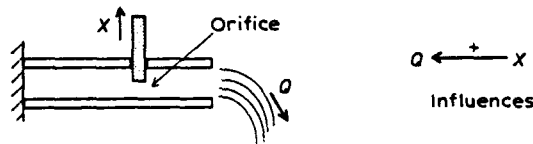
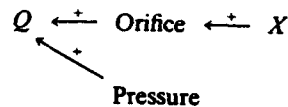


Figure 2-1: A simplified water tap

At a more detailed level, the influences correspond to standard physical laws and effects [21]. The tap's detailed behavior, as shown below, is composed of the following influences: (1) As the

⁵The domain within which the proposed transformational approach has been investigated is a class of mechanical devices which alter the physical properties of input materials upon being activated by either an external or an internal mechanical signal. Such devices control the parameters of given processes. An assumption we make is that the process already exists. We do not consider devices which create processes. For example, in a water tap it is implicitly assumed that liquid flow is already present. We do not reason about the process itself [10], but about devices that use the processes. Based on these assumptions, devices can be viewed as black-boxes which take inputs and produce desired outputs. In the physical domain, three types of inputs and outputs have been identified: signals, energy and materials [48].

gate is opened, its position (X) influences the *Orifice* size and, (2) the total flow of water (Q) is influenced by the *Orifice* size and the *Pressure* difference. The first influence is based on rigid-body motion and follows from the definition of an orifice. The second set of influences are based on Bernoulli's theorem.



Sets of qualitative influences can be combined to capture the behavior of more complex devices. The see-saw shown in Figure 2-2 has three major behavioral parameters: Ω , the angular position of the see-saw and the positions of the two ends of the see-saw ($X1$ and $X2$). The main influences are $X1 \xrightarrow{+} \Omega$, $X2 \xrightarrow{-} \Omega$ and $X2 \xrightarrow{-} X1$. These influences form a directed graph.

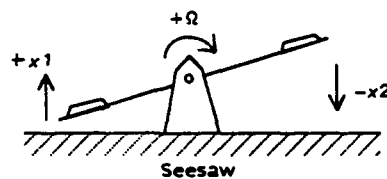


Figure 2-2: A see-saw

2.4.2 Total Influences

Quantities can have multiple influences on one another. If there are multiple paths from some node B to a node A , and if all the paths are positive (or negative), then the total influence of B on A is positive (or negative).

If, however, the paths are not all of the same sign, then the total influence can be either positive, negative or zero. This depends on the actual magnitudes of the influences. As CADET does not, as yet, reason about magnitudes, the decision on the final outcome is left to the designer. In design, it is sometimes useful to have zero total influence by making positive and negative influences cancel one another. This is different from the non-existence of an influence between quantities. For example, if there are multiple paths from B to A , and if there is at least one path with an influence of opposite sign to the other paths, then B could be made independent of A by proportionately adjusting the magnitudes of the influences.

Let's re-consider the hot-cold water faucet. The behavior of this device can be viewed as the combination of known behaviors. For example, the hot-cold water faucet's behavior can be represented as shown below (Figure 2-3). Two input signals St and Sm control the mix temperature and the mix flow-rate. When St is increased, then the total quantity of hot water Qh increases. At the same time, due to the see-saw principle, the signal $St1$ decreases, causing the total quantity of cold water to proportionately decrease. Hence, as shown in the influence diagram, when St increases, the mix temperature Tm increases. The two influences are: $Tm \leftarrow^{+} Qh \leftarrow^{+} X1 \leftarrow^{+} St$ and $Tm \leftarrow^{-} Qc \leftarrow^{+} X2 \leftarrow^{+} St1 \leftarrow^{-} St$. The total influence of St on Tm on both paths is positive.

As St is a temperature control signal, it is not supposed to change the total flow rate (Qm) while the temperature of the mix (Tm) is changed. In the influence graph, the total influence of St on the mix flow-rate Qm is zero. This is done by canceling the total positive and negative influences. As shown in the influence graph, there are two paths from St to Qm : $Qm \leftarrow^{+} Qc \leftarrow^{+} X2 \leftarrow^{+} St1 \leftarrow^{-} St$ and $Qm \leftarrow^{+} Qh \leftarrow^{+} X1 \leftarrow^{+} St$. The first path has a total negative influence and the second path has a total positive influence. By adjusting the magnitudes of the influences, we can achieve total zero influence. Physically this implies that the see-saw's fulcrum is placed exactly in between the two taps.

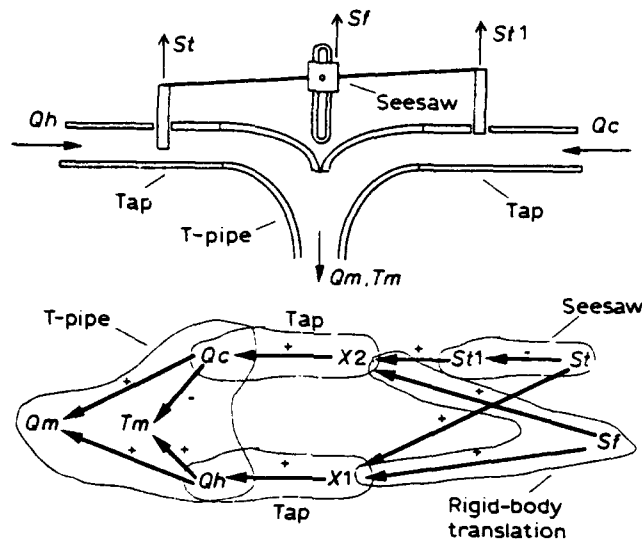


Figure 2-3: Faucet's Influence Graph

The notion of influence graphs is a very general one. It applies to any domain in which behavior can be characterized as a set of quantities that relate to one-another. Consider an example from the labor relations domain. We know that a company's profitability is influenced by the market, product quality, and productivity. In turn, productivity is influenced by technology and labor morale. Further, a worker's morale may be influenced by salary, benefits and job-security [61]. Given such an influence graph, it is possible to predict possible outcomes of given perturbations.

In CADET, influence graphs are used to represent the behavior of devices. When this representation is incorporated in a device case base, it becomes possible to retrieve cases which match given behavior specifications. If retrieval using the design specification fails to retrieve relevant cases, the system should be able to recognize how a combination of component behaviors could produce the required effect.

2.5 Index Transformation

Most existing case-based systems use a pre-defined set of indices to access cases in memory. This indexing strategy is limiting since salient features of the current case which could constitute good indices may not directly match the pre-defined index set. Index transformation is a way to change the given salient features of the current problem to match the indices under which previous cases have been stored, thus making accessible to the problem solver previously inaccessible cases. The transformation technique described here is applicable to any domain in which behavior can be modeled as a graph of influences.

We will now examine two rules which are used to transform given goals into more elaborate sets of influences that are behaviorally equivalent to the goal. The hypothesis is that, if one cannot find a case relevant to a given goal, then it might be possible to find several cases or parts of

cases that are relevant to elaborations of the given goal.

2.5.1 Two Rules for Reasoning about Influences

1. *The function of function (chain) theorem.* Let $u = f(x, y)$, where x and y are independent quantitative variables with respect to u . Let $z = g(u, v)$ be another quantitative function. Then,

$$\left[\frac{\partial z}{\partial x} \right] = \left[\frac{\partial z}{\partial u} \right] \cdot \left[\frac{\partial u}{\partial x} \right]$$

Where, the square brackets ($[]$) indicate a qualitative operator. The operator returns +, - or 0 when the numerical value of the expression within the brackets is more than, less than or equal to zero respectively. The arithmetic operators are also qualitative.

2. *The total influence theorem.* Let $z = f(p, q)$ be a continuous quantitative function. Then the total influence

$$[\Delta z] = \left[\frac{\partial z}{\partial p} \right] \times [\Delta p] + \left[\frac{\partial z}{\partial q} \right] \times [\Delta q]$$

The total influence gives the net qualitative increment in the dependent variable as the independent variables are changed.

Two analysis rules arise directly from these theorems (Proofs in Appendix B):

Analysis Rule 1. If some quantity x is known to influence u and if u is known to influence z then one can infer that x influences z .

Analysis Rule 2. If p and q are known to influence z and if x is known to influence p and q , then one can infer that x influences z .

The two analysis rules can be applied to influence graphs to derive new influences. Repeated application of these rules will generate new influences that exist in a given system, but may not have been explicit. For example, consider a kitchen sink with two taps pouring water into it (Figure 2-4). The flow rates of the two streams of water are $Q1$ and $Q2$ and the depth of water in the sink (at equilibrium) is D . The influences on D are: $D \xleftarrow{Op1} Q1$, $D \xleftarrow{Op2} Q2$, where $Op1$ and $Op2$ are both positive influences. The rate of flow out of the sink ($Q3$) is influenced by the depth of water D : $Q3 \xleftarrow{Op3} D$ where, $Op3$ is positive.

From the first analysis rule we can conclude that the input flow $Q1$ will influence the total outflow $Q3$ as $Q3 \xleftarrow{Op1 \times Op3} Q1$. In other words, an increase in $Q1$ will cause the rate of flow through the drain $Q3$ to increase. The sign of the influence is the qualitative multiplication of $Op1$ and $Op3$.

The analysis rules can be used to propagate influences and analyze the behavior of systems as shown above. A detailed description of how qualitative methods may be used to perform simulations is presented in [34]. We are, however, interested in using the rules for design, rather

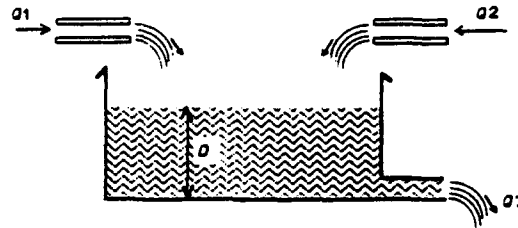


Figure 2-4: Kitchen Sink with two flow inputs

than analysis. We do this by applying the two rules in reverse. The intuition is that design is the inverse of analysis. In analysis, the artifact is given and one has to derive the behavior. In design, on the other hand, the desired behavior is given and one has to come up with the artifact. Here are the two design rules:

Design Rule 1. If the goal is to have x influence z , and if it is known *a priori* that u influences z , then the goal could be achieved by making x influence u .

Design Rule 2. If the goal is to have x influence z and if it is known *a priori* that some two quantities p and q influence z then, the goal could be achieved by making x influence p or q , or both.

The two design rules transform a given influence into a more detailed set of influences which are behaviorally equivalent to the original influence. As transformation operators, the first rule yields a serial transformation and the second one yields parallel transformations.

Before embarking on the details of how the design rules are used, here is an overview of how the rules are used in behavioral synthesis:

1. The initial behavior specification (the goal) is used as an index into the case base. Exact matching is first attempted. An exact match occurs when the variables and the sign of the influences are the same. If such a match is not found, then a partial match is attempted.
2. If no cases are retrieved, the goal is transformed using the two design rules. This is first done using the given domain principles.
3. If constraints on the behavior have been specified, then the generated transforms are checked. Violating transforms are pruned off the search tree.

4. The transformed influences are used as indices to find matching cases or parts of cases in the case-base.
5. If no cases are found, or if only part of the goal influences are found to match cases, the goal influences are elaborated by hypothesizing influences.
6. Once again case retrieval is attempted. This time, however, hypothesized variables have to be bound to the variables in the cases.
7. If all the influences in a transformed index are not matched to cases, further transformations are attempted. This step is controlled by the user. At any stage, the user may stop the transformations and try to fill in the gaps from his or her own experience.

2.6 Using the design rules

The design rules are used to transform goal influences into more elaborate influence graphs. In CADET, the transformation is done in two ways: (1) by using domain laws and, (2) by hypothesizing new variables.

2.6.1 Using Domain Laws

The influences implied by domain laws may be used to elaborate given goals. For example, assume it is our goal to achieve the influence: $z \leftarrow x$, also assume that there are no known designs that can achieve this effect directly. If, however, there is some domain principle which states that a quantity u influences z , then the goal may be achieved by having x influence u . The goal is hence elaborated to: $z \leftarrow u \leftarrow x$. This new influence graph is used as a new index into the case base. If cases or part of cases with influences that match the goal are found, they are retrieved and used.

Reconsider the kitchen sink example (See Figure 2-4). Let's say we want to design something which will allow us to control the total outflow $Q3$ with respect to some external physical signal: Sig . The required (goal) influence is $Q3 \leftarrow^{+} Sig$. This influence can be transformed by finding what factors influence $Q3$. From the *Law of Conservation of Mass* we know that: $Q3 = Q1 + Q2$ at equilibrium. Using symbolic calculus the following influences are derived from the law: $Q3 \leftarrow^{+} Q1$, $Q3 \leftarrow^{+} Q2$. Influences between $Q1$ and $Q2$ are not used as they are specified to be independent variables. It follows that the variable Sig can be made to influence $Q1$ or $Q2$, and hence influence $Q3$ indirectly. There are three possible influences (a) the original influence $Q3 \leftarrow^{+} Sig$, (b) the first indirect influence: $Q3 \leftarrow^{+} Q1$, $Q1 \leftarrow^{+} Sig$, and (c) the second indirect influence: $Q3 \leftarrow^{+} Q2$, $Q2 \leftarrow^{+} Sig$. Combinations of the above influences yield possible transformations. Combinations include selecting any one influence, any two at a time or all three. This gives us seven combinations including the original influence. These elaborations can be used to retrieve cases.

Let's pick the influences $Q3 \leftarrow^{+} Q1$ and $Q1 \leftarrow^{+} Sig$. The first influence need not be

designed for as it is already part of the given kitchen sink. The second influence matches the influence $Q \leftarrow^+ X$ in the tap case (Figure 2-1). The nodes Q and $Q1$ match because they are both flow rates and X matches Sig because they are both physical signals. The matching code uses object and concept class hierarchies to perform such matches. After the tap is retrieved, it may be used to yield the design in Figure 2-5.

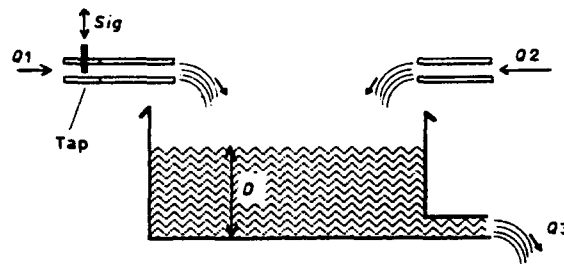


Figure 2-5: Kitchen Sink with a Tap

2.6.2 Hypothesizing new variables

If the given domain laws are unable to find elaborations that can be realized by the cases in memory, one can try to hypothesize variables. The idea is to hypothesize new influences and then find cases which may be used to achieve those influences. For example, the goal $z \leftarrow x$ may be elaborated to $z \leftarrow Var1 \leftarrow x$ using design rule 1. A new variable $Var1$ is hypothesized as an intermediary. The elaboration is then used to find cases in memory. This time however, the system looks for two influences which match the goal and bind the unknown variable $Var1$.

As new variables are introduced, corresponding new influences are hypothesized. In addition, as influences are all supposed to be based on physical laws or principles, the introduction of new variables implies that laws or principles, unknown to the program, are being hypothesized. After hypothesizing influences, the case base is used to find prior designs which may embody some physical law or principle that matches the hypothesized influence. With this approach one often retrieves cases from outside the current design domain that are analogically related to the current design problem. It is for this reason that CADET's solutions are innovative⁶. The approach is able to solve design problems by drawing analogies to prior designs and by exploiting physical

⁶For a definition of innovation refer [44]

laws and effects other than those included in the given domain description.

Consider the design of a device which controls the flow of water into a flush tank. The initial configuration is shown in Figure 2-6: a pipe is delivering some water to a tank which fills over time. The behavior of the device to be designed can be specified as follows: *as the depth of water (D) in the tank increases, the rate of flow of water into the tank (Q) should decrease.* The influence is given by $Q \leftarrow^- D$. This influence may be used as an index to retrieve cases with. If a matching case is not found, then one would have to consider using a combination of cases which are behaviorally equivalent to the specified behavior. Assume also that no relevant domain principles have been identified *a priori*.

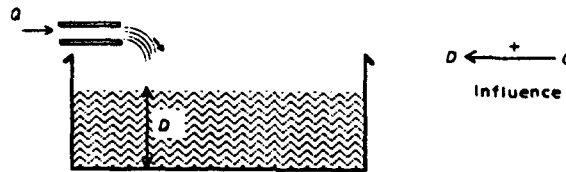


Figure 2-6: Initial Configuration before Design begins.

Two serial transformations (by design rule 1) of the influence, $Q \leftarrow^- D$ yields the following three sets of influences:

1. $Q \leftarrow^- \text{Var2} \leftarrow^+ \text{Var1} \leftarrow^+ D$
2. $Q \leftarrow^+ \text{Var2} \leftarrow^- \text{Var1} \leftarrow^+ D$
3. $Q \leftarrow^+ \text{Var2} \leftarrow^+ \text{Var1} \leftarrow^- D$

Consider the second set of influences. The influence $\text{Var2} \leftarrow^- \text{Var1}$ can be matched to the see-saw influence $X2 \leftarrow^- X1$ which binds the two unknown variables. The influence $Q \leftarrow^+ X2$ matches the tap (Figure 2-1). The remaining influence $X1 \leftarrow^+ D$ matches a float (Figure 2-7).

The resulting design is shown in Figure 2-8. The figure has two parts: a synthesis strategy and the actual synthesized design. Currently, CADET only generates a synthesis strategy. It outputs a list of cases or case pieces with information on how they should be connected. We are currently addressing issues in physical synthesis such as, adaptation of case pieces, recognition and resolution of physical interactions, and the management of undesirable side-effects. For

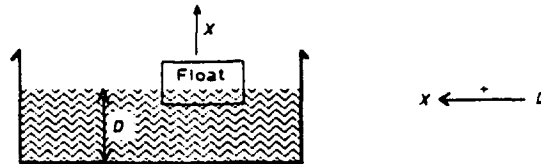


Figure 2-7: A simple float

example, the see-saw (as retrieved from case memory) may actually be 10 feet long and would have to be appropriately re-sized to suit the flush tank design. Further, the position of the fulcrum has to be moved to match the vertical distances moved by the float and the tap's plunger.

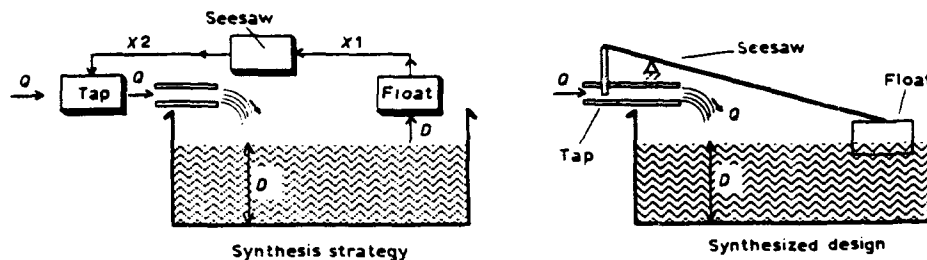


Figure 2-8: A flush tank

Let's look at another possible design configuration for this example. The third set of influences: $Q \leftarrow^+ Var2 \leftarrow^+ Var1 \leftarrow^- D$ can generate alternative design configurations. The first influence $Var1 \leftarrow^- D$, says that as the water level increases, some quantity $Var1$ decreases. An ultrasonic distance measuring device, held over the water surface, could provide this behavior. The output of this device is an electrical signal. Let's call it Sig and bind it to $Var1$. The

influence $Q \leftarrow^+ Var2$ matches a basic tap by binding $Var2$ to X (which is a linear movement). Finally, we are left with the influence $X \leftarrow^+ Sig$ which says that when an electrical signal increases a body moves linearly in the X direction. A positioning device with a linear ratchet and motor can provide this function. The resulting design is shown in Figure 2-9.

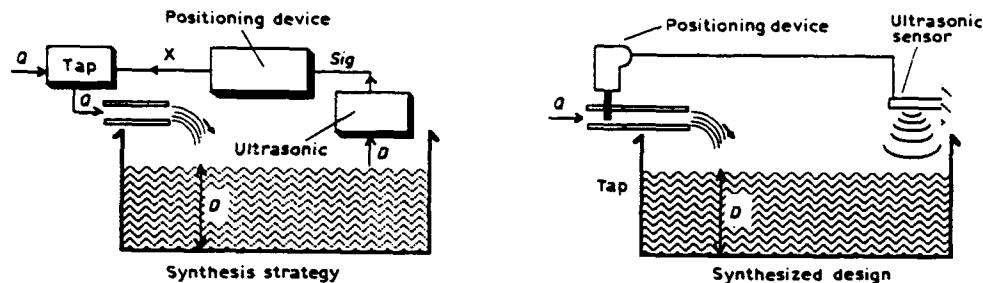


Figure 2-9: A flush tank: exploiting extra-domain principles

Through the process of influence hypothesis and matching, the system is able to use physical laws and principles embedded in prior design cases to achieve its current goals. In this way, CADET is opportunistic about the principles it exploits in a design. This is unlike other approaches which assume that all the relevant principles have been identified *a priori* [79]. Because CADET hypothesizes influences, it does not limit itself to the given knowledge. Consequently, it can generate elaborations that represent designs that have never been conceived of before.

2.7 Case Matching

Issues relating to matching indices to case attributes has been an important part of CBR research [26, 51]. These efforts have concentrated on retrieving cases using indices that match specific attributes about cases. In CADET, the case matching problem includes matching graphs of influences.

Behavior matching is done in several ways:

1. **Exact Match.** The quantities (nodes) and influences (arcs) match exactly.
2. **Abstract Match.** Nodes are matched using object and concept hierarchies. For example, the translatory-signal X will match a rotational-signal Ω as they both are physical-signals.
3. **Structure Matching with Variable Binding.** When CADET hypothesizes variables and influences, they have to be matched against cases to find appropriate

bindings for the unknown variables. The system looks for common sub-graphs in the case and the index. In this way it is able to find relevant sub-behaviors embedded in the cases.

Consider the flush tank example. The elaborated index $Q \leftarrow^+ Var2 \leftarrow^- Var1 \leftarrow^+ D$ could be matched against the faucet to extract the see-saw and tap assembly. The match, shown in Figure 2-10, shows the recognition of common "sub-behaviors" between an elaboration and a design case.

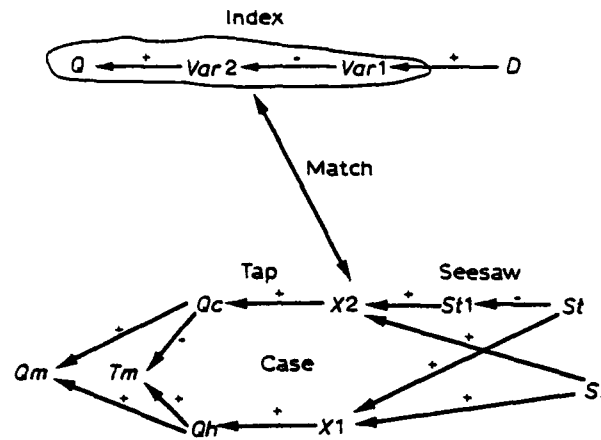


Figure 2-10: Matching "sub-behaviors"

The match involves the binding the $Var1$ and $Var2$, and is done at different levels (see below). The nodes Q and Q_c match because they are both liquid-flow-rates.

4. **Matching at different levels of detail**. We have to be able to match cases which may not be at the same level of abstraction as the index.

For example, the index $Q1 \leftarrow^+ X1$ will match the tap case which has the following behavior: $Q \leftarrow^+ X$. Both index and case are at the same level of detail and the matching is straightforward. If, however, the tap case was represented at a more detailed level: $Q \leftarrow^+ Orifice-size \leftarrow^+ X$, then the given index $Q1 \leftarrow^+ X1$ will not match. We don't have a good solution to this problem. In CADET this problem is solved by waiting for the next elaboration of the index. If the index is elaborated by the first hypothesizing design rule, it becomes: $Q1 \leftarrow^+ Var1 \leftarrow^+ X1$. This elaborated index will match the case.

2.8 Constraint Checking

The search for elaborations is controlled by checking for constraint violations. We have identified two types of constraints: positive and negative. Positive constraints are those that should necessarily hold good in the final design. These constraints are typically included in the goal statement. Negative constraints are those influences which should not exist in the final design. This is denoted by: $A \leftarrow^- B$. There are two ways of checking for the non-existence of

an influence: (1) No direct influence, and (2) No total influence.

No direct influence. Some quantity B is said to not influence quantity A if, after exhaustively applying the two analysis rules, there is no direct or indirect direct path of influences from B to A .

No total influence. This is a more subtle concept. In design, it is sometimes possible to make one quantity have no apparent influence on another by making positive and negative influences cancel each other. For example, if there are two paths of opposite signs from B to A , then by proportionately adjusting the influences, (depending on quantitative magnitudes), it may be possible to make B 's total influence on A be zero. Appendix A (extended design example) illustrates the use of this type of constraint checking.

2.9 Concluding Remarks

Problem solving in the domain of Engineering Design imposes a set of requirements on a problem solver: (a) the representation needs to capture and integrate several levels of abstraction from the linguistic to the physical, incorporating linguistic specifications, laws of physics, and constraints, (b) the problem solver should be able to reason both symbolically and analytically at different problem solving stages and integrate the process and results of its reasoning, (c) verification techniques should be incorporated in the problem solving.

We have presented an approach to the conceptual design of hydro-mechanical systems using a case base of previous designs that realize subfunctions of the desired artifact. The process consists of applying behavior-preserving transformations, based on a qualitative calculus, to an abstract description of the desired behavior of the device until a description is found that closely corresponds to some collection of relevant cases. The major benefits of the approach are: (a) it allows for retrieval of relevant cases without imposing a predetermined decomposition of the design, (b) it is a generative approach that utilizes knowledge of domain laws, design principles, such as simplicity, and behavioral constraints to reason from design goals to possible solution structures, (c) it can identify "missing" cases, necessary for completion of the design, and (d) the resulting transformations are guaranteed to be behaviorally equivalent to the original specification.

APPENDIX A. Conceptual Design Example

In this appendix, we will examine an example that illustrates the workings of the CADET system. The example is about the design of a hot/cold water faucet. The function of the faucet is described as:

A device which *mixes* hot water at temperature T_h and cold water at temperature T_c with flow rates Q_h and Q_c respectively and allows the control of the mixed water temperature T_m by a mechanical signal S_t and its flow rate Q_m and by a mechanical signal S_f . In addition, the two controls should be independent: S_t should not influence Q_m and S_f should not influence T_m .

This specification is input to CADET as goals, constraints and a qualitative description of the governing physical laws and principles. The program then starts case matching and elaboration. It stops when it finds the first set of cases or case pieces that fully match an elaboration. It finally draws the synthesis strategy diagram on a graphics window. If the user is not satisfied with the design, he/she may edit the specification file, add new constraints and restart the system.

Goals.

The goals can be represented using influences such as:

$$\left[\frac{\partial T_m}{\partial S_t}\right] = [+]$$

$$\left[\frac{\partial Q_m}{\partial S_f}\right] = [+]$$

Here we have interpreted "control" as the influence value [+]. This is only a convention. The influence value could also have been [-]. This does not affect our reasoning procedure since S_t and S_f are unknowns.

Constraints.

For the faucet, the behavior specification says that the signal S_t should not influence Q_m and the signal S_f should not affect T_m . We may try to represent these as positive constraints as:

$$\left[\frac{\partial T_m}{\partial S_f}\right] = [0], \quad \left[\frac{\partial Q_m}{\partial S_t}\right] = [0]$$

However, this formulation implicitly assume here that T_m *depends* on S_f . This need not be true at all since, T_m might be totally independent of S_f . To avoid this ambiguity, we make use of negative constraints. For CADET the above constraints are input as:

$$\begin{aligned} \left[\frac{\partial T_m}{\partial S_f}\right] &\neq [+], \quad \left[\frac{\partial T_m}{\partial S_f}\right] \neq [-] \\ \left[\frac{\partial Q_m}{\partial S_t}\right] &\neq [+], \quad \left[\frac{\partial Q_m}{\partial S_t}\right] \neq [-] \end{aligned}$$

During constraint checking, if any of these influences occur as *effects* of the device behavior, the alternative is discarded.

Process description.

The physical laws and principles governing the process are also input to CADET. For the faucet the process controlled by the device is the mixture of fluid flow. The behavior of the mixture can be described in terms of the following two equations:

$$\begin{aligned} Q_m T_m &= Q_h T_h + Q_c T_c && \text{Conservation of Energy} \\ Q_m &= Q_c + Q_h && \text{Conservation of Mass} \\ \text{where: } T_h &> T_c \end{aligned}$$

The following set of influences describing the process are derived from the above principles:

$$\left[\frac{\partial T_m}{\partial T_c}\right]=[+], \quad \left[\frac{\partial T_m}{\partial T_h}\right]=[+], \quad \left[\frac{\partial T_m}{\partial Q_c}\right]=[-], \quad \left[\frac{\partial T_m}{\partial Q_h}\right]=[+], \quad \left[\frac{\partial Q_m}{\partial Q_c}\right]=[+], \quad \left[\frac{\partial Q_m}{\partial Q_h}\right]=[+]$$

2.9.1 The Design Process

CADET first tries to retrieve cases which match the given specifications. If it cannot find a matching case, it starts elaborating the given goal. CADET first applies the serial operator (inverse application of the function of function rule). The following eight alternatives are generated⁷:

1. $(T_c S_t +) (Q_c S_f +)$
2. $(T_c S_t +) (Q_h S_f +)$
3. $(Q_c S_t -) (Q_c S_f +)$
4. $(Q_c S_t -) (Q_h S_f +)$
5. $(T_h S_t +) (Q_c S_f +)$
6. $(T_h S_t +) (Q_h S_f +)$
7. $(Q_h S_t +) (Q_c S_f +)$
8. $(Q_h S_t +) (Q_h S_f +)$

These alternatives are combinations of the influences of the two input signals. They are checked for constraint satisfaction using the two analysis rules. The idea is to find all the influences implied by the design alternative and to then look for influences that violate constraints. For

⁷In CADET the influence $\left[\frac{\partial Q_m}{\partial Q_h}\right]=[+]$ is represented as the list $(Q_m Q_h +)$.

instance, the first alternative is checked as shown in Figure A-11. The two clauses of the alternative are combined with the other known influences to derive new influences. Using the first analysis rule, the clause $(T_c S_t +)$ is combined with the known influence $(T_m T_c +)$ to infer that $(T_m S_t +)$. As shown in the figure, making such inferences can help find violations.

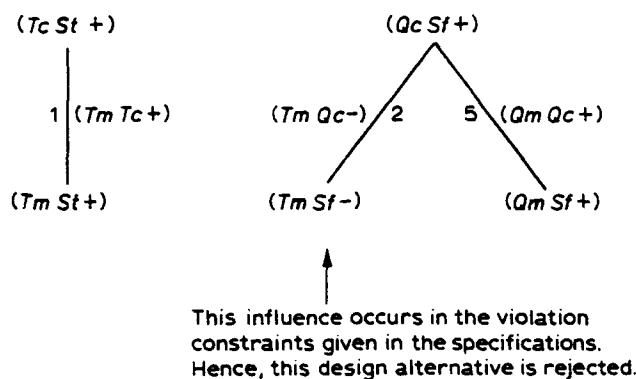


Figure A-11: Constraint propagation for serial design alternative 1

In this example, the first eight alternatives are all found to violate the given constraints. This however, does not mean that further elaborations of the alternatives will also fail to satisfy the constraints. The reason is that these alternatives represent too strong a coupling among device parameters. Let's see what this means.

If CADET is told to ignore the constraints, then the program accepts the above eight designs and retrieves the following unique sets of cases:

1. (Water-heater) (Tap)
2. ((Tap)(See-saw)) (Tap)
3. (Water-heater) (Tap)
4. (Tap) (Tap)

Alternatives 1 and 4 are sketched (manually) in Figure A-12 and Figure A-13 respectively. Both designs violate the independence constraints. The effects of S_t and S_f are coupled. We have to decouple them through further elaborations of the index.

CADET continues by applying the second design rule. It generates 28 distinct design alternatives. Some of them are:

1. $(T_c S_t +) (Q_c S_f +) (Q_c S_t -) (Q_c S_f +)$
2. $(T_c S_t +) (Q_c S_f +) (Q_c S_t -) (Q_h S_f +)$
3. $(Q_h S_t +) (Q_h S_f +) (T_h S_t +) (Q_c S_f +)$
4. $(Q_c S_t -) (Q_h S_f +) (Q_h S_t +) (Q_c S_f +)$
5. $(T_c S_t +) (Q_c S_f +) (T_c S_t +) (Q_h S_f +)$

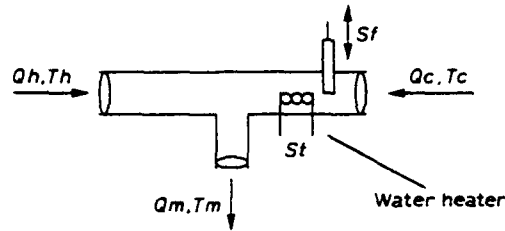


Figure A-12: Potential final design for serial design alternative 1.

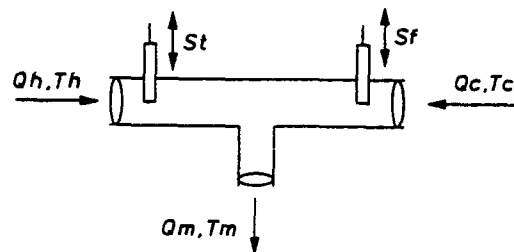


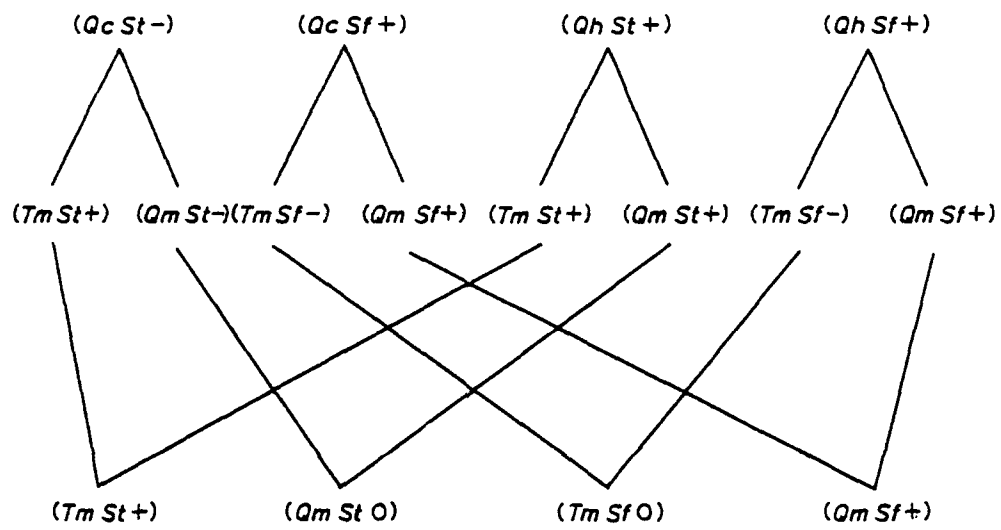
Figure A-13: Potential final design for serial design alternative 4.

The first three are rejected based on constraint checking as done before. The fourth alternative produces the following influences as the end result of exhaustive influence propagation which has to be done for constraint checking. (Figure A-14):

$$(Q_m S_t -) (T_m S_t +) (Q_m S_f +) (T_m S_f +) (Q_m S_t +) (T_m S_t +) (Q_m S_f +) (T_m S_f -)$$

In this influence graph, S_t influences T_m positively on the whole but influences Q_m both negatively *and* positively. By applying the total influence property we see that S_t will not influence Q_m if the quantitative increase and decrease are equal. The interpretation is that the design *could* potentially satisfy the constraints. Similarly, we find that S_f will influence Q_m

Figure A-14: Constraint Propagation for a parallel design alternative



positively, can be made to not influence T_m (total influence). The potential design, based on the cases retrieved for this is given in Figure A-15.

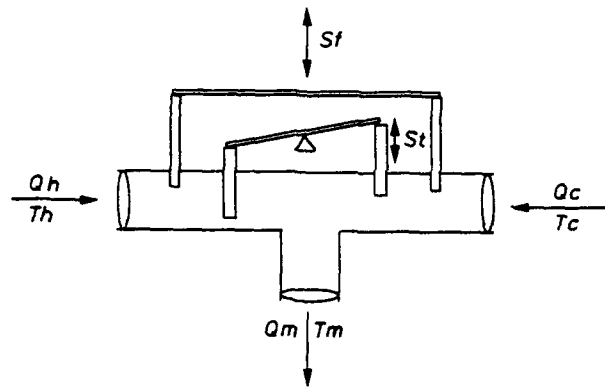


Figure A-15: Potential final design for parallel design alternative 4.

Another feasible parallel design alternative generated by CADET is: $(T_c S_t +) (Q_c S_f +) (T_c S_t +) (Q_h S_f +)$. and the potential final design for this alternative is:

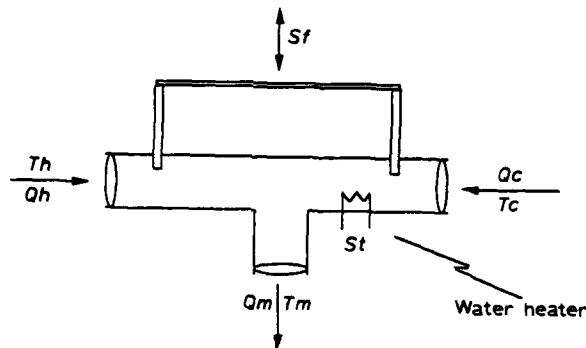


Figure A-16: Potential final design for parallel design alternative 5.

APPENDIX B. Proof of the Analysis Theorems

Proofs of the two rules are based on an assumption of continuity [6] and differentiability.

1. *The Function of Function rule.* Let $[u] = f([x], [y])$, where $[x]$ and $[y]$ are independent variables with respect to $[u]$. Let $[z] = g([u], [v])$ be another qualitative function. Then,

$$\frac{\partial [z]}{\partial [x]} = \frac{\partial [z]}{\partial [u]} \cdot \frac{\partial [u]}{\partial [x]}$$

Proof:- Let $[x]$ and $[y]$ change by $\Delta[x]$ and $\Delta[y]$. Let the corresponding increments in $[u]$ and $[z]$ be $\Delta[u]$ and $\Delta[z]$ respectively. Then, we can write (assuming continuity and differentiability)

$$\frac{\Delta[z]}{\Delta[x]} = \frac{\Delta[z]}{\Delta[u]} \cdot \frac{\Delta[u]}{\Delta[x]}$$

When $\Delta[x] \rightarrow [0]$, then, since $\Delta[u] \rightarrow [0]$ as $\Delta[x] \rightarrow [0]$, we have,

$$\frac{\partial [z]}{\partial [x]} = \lim_{\Delta[u] \rightarrow [0]} \frac{\Delta[z]}{\Delta[u]} \cdot \lim_{\Delta[x] \rightarrow [0]} \frac{\Delta[u]}{\Delta[x]}$$

or,

$$\frac{\partial [z]}{\partial [x]} = \frac{\partial [z]}{\partial [u]} \cdot \frac{\partial [u]}{\partial [x]}$$

Q.E.D.

The notion of independence among variables in the system is crucial for the valid application of the function of function theorem. Following, are some useful definitions of dependence and independence of variables:

1. A variable y is said to be *relatively dependent* on another variable x if its value is a function of x . In such a context $[x]$ is said to be *relatively independent* of $[y]$.
2. A variable is defined as an *absolute independent variable* if this variable is not relatively dependent on (or not influenced by) any other variable in the system. From a design point of view, we can say *all* input variables are *absolute independent* variables.
3. A variable is defined as an *absolute dependent variable* if no other variable in the system is relatively dependent on this variable. All output variables possess absolute dependence.
4. A variable that is neither an absolute independent variable nor an absolute dependent variable is an *intermediate variable*. An intermediate variable can be either a relative dependent variable or a relative independent variable depending on the context.
5. The function of function theorem requires only relative dependence and independence.

Interestingly, the same problem occurs in numerical calculus. Consider the following algebraic equation: $y = 3x + 4z$ where $\partial y/\partial x = 3$, and $\partial y/\partial z = 4$, from which we can calculate for $\partial(z)/\partial(x)$ as $\partial(z)/\partial(y) \times \partial(y)/\partial(x) = 1/4 \times 3 = 3/4$. But, from the original equation we can write: $z = y/4 - 3x/4$ from which we get $\partial z/\partial x = -3/4$, which contradicts what was just calculated. The fallacy is that, while using the function of function rule we violated the necessary conditions that z is not a dependent variable with respect to x . When we calculated $\partial y/\partial x = 3$, we assumed that $\partial z/\partial x = 0$, but later derived a relation between z and x leading to a contradiction.

2. The total differential Theorem. Let $[u] = f([x],[y])$ be a continuous qualitative function. Then the total differential

$$\Delta[u] = \frac{\partial[u]}{\partial[x]} \times \Delta[x] + \frac{\partial[u]}{\partial[y]} \times \Delta[y]$$

The total differential gives the net increment in the dependent variable as the independent variables are changed.

Proof :- The total increment in $[u]$ can be written as

$$\begin{aligned} \Delta[u] &= f([x]+\Delta[x],[y]+\Delta[y]) - f([x],[y]) \\ \Rightarrow \Delta[u] &= f([x]+\Delta[x],[y]+\Delta[y]) - f([x],[y]+\Delta[y]) + f([x],[y]+\Delta[y]) - f([x],[y]) \end{aligned}$$

which can be written as :

$$\Delta[u] = \frac{f([x]+\Delta[x],[y]+\Delta[y]) - f([x],[y]+\Delta[y])}{\Delta[x]} \cdot \Delta[x] + \frac{f([x],[y]+\Delta[y]) - f([x],[y])}{\Delta[y]} \cdot \Delta[y]$$

Now as $\Delta[x]$ and $\Delta[y] \rightarrow [0]$, we have in the limit

$$d[u] = \frac{\partial[u]}{\partial[x]} \cdot d[x] + \frac{\partial[u]}{\partial[y]} \cdot d[y] \quad \text{Q.E.D}$$

Chapter 3

The CADET system

CADET's approach to design has three main characteristics, (a) transformation of high level functional specifications to a set of alternative structural descriptions that satisfy the given specifications, (b) adaptation of retrieved previous designs and design pieces to fit specifications and constraints in the current design problem, and (c) synthesis of the most promising alternative(s) from components whose combined behavior is equivalent to the overall device specifications. The input to CADET is the set of design specifications and its output is a conceptual schematic that meets the given specifications. The input design specifications consist of functional and behavioral descriptions and physical constraints. Functional specifications describe the interactions of the device with its environment (e.g., its use). Behavioral specifications describe the behavior enabled by the structural configuration of the device. For example, the functional specification of a clock is to "tell time", whereas its behavior is the movement of its hands on the dial caused by the movement of the gears (the structural parts) inside the clock. Physical constraints describe constraints on the physically realizable structural description of the device. In particular, the functional description in CADET consists of functional goals, functional constraints, and a functional process description. The behavioral description consists of behavioral goals, behavioral constraints and a behavioral process description. The functional goal of a clock is "to tell time", a functional constraint could be "usable by a blind person", a functional process description could be "the telling of time is through touch or sound". A behavioral goal for a clock could be "find a mapping of a day to some finite- numbered set of symbols and exhibit the mapping with an indication of the correspondence of current time to one of the symbols", a behavioral constraint could be "find the minimum number of needed symbols", and a behavioral process description could be "the indication of the correspondence of current time to one of the symbol should be periodic".

CADET has access to a case memory and engineering domain laws and principles. Each case is represented in terms of a multi-layered representation expressing function, behavior and structure of the desired device and relations among them. The representational abstractions that are used for indexing the case memory are: (1) Linguistic descriptions, (2) Functional Block Diagramming, (3) Qualitative influence graphs, (4) Qualitative States (5) Structural features (including device topology, material, weight and dimensions), and (6) Performance features (including cost, reliability, availability, device output values under different conditions).

CADET's case based design process consists of the following steps that are iteratively applied as new subgoals are generated during problem solving [68]. Though we present these steps sequentially, they may be interleaved during problem solving.

1. *Development of a Functional Description.* At the simplest level, the desired artifact can be viewed as a black-box which takes certain inputs and produces desired outputs. The function of the black-box is described by qualitative relations explaining how the inputs and outputs are related. The system's job is to produce the structural description of a physically realizable artifact which will convert the inputs into the desired outputs.
2. *Retrieval of Cases.* A set of design cases (or case parts) bearing similarity to a given collection of features are accessed and retrieved. Retrieval is performed using not only the existing features of the input specification, but also indices arising from index transformation strategies.
3. *Case Adaptation to fit specifications.* Retrieved cases and case pieces must be compared to input specifications, so that deviations from desired specifications can be identified and appropriate adaptations can be generated.
4. *Development of a Synthesis Strategy.* A synthesis strategy is a description of how the various cases and case pieces will fit together to yield a design that meets the given specifications.
5. *Physical Synthesis.* Realization of the synthesis strategy at a physical level. This is a difficult problem since undesirable side effects among case parts may occur.
6. *Verification.* Adverse interactions could lead to non-conformance of the design to the desired specifications. This is verified through quantitative and qualitative simulation. If the simulation is correct, and if all the constraints are satisfied, then the design is successful. If not, repair (next step) is attempted.
7. *Debugging.* Debugging involves the process of asking relevant questions based on a causal explanation of the bug. These questions serve as cues into memory to retrieve cases about bugs and their repairs.

The current implementation of CADET, discussed in this paper, covers the first five steps indicated above. Verification and debugging is currently left to the human designer. Our system's usefulness lies in it's ability to access a large database of prior designs and find relevant components and alternative configurations. CADET is aimed at being a designer's brainstorming assistant, not his replacement.

3.10 CADET System Architecture

CADET's architecture is centered around a Design Blackboard (Figure 3-17). The blackboard is used to maintain information about the various design alternatives that are being actively considered at any given time. Our choice of a blackboard architecture was motivated by several considerations. First, the blackboard provides continuous visibility of the current status of the design, so it is easier for the user to make informed interactive decisions. Second, the blackboard supports communication and cooperation among the various subsystems (e.g., case base, materials module, constraint checking module). Third, the blackboard provides control

mechanisms, such as posting of events and checking of dependencies that are helpful in detecting conflicts and guiding the evolution of the design in the right direction. Fourth, since design is in general an underspecified task, the blackboard provides flexible means for asserting the dynamically arising new goals and constraints, thus allowing the relevant design modules to be promptly informed and engage in appropriate problem solving.

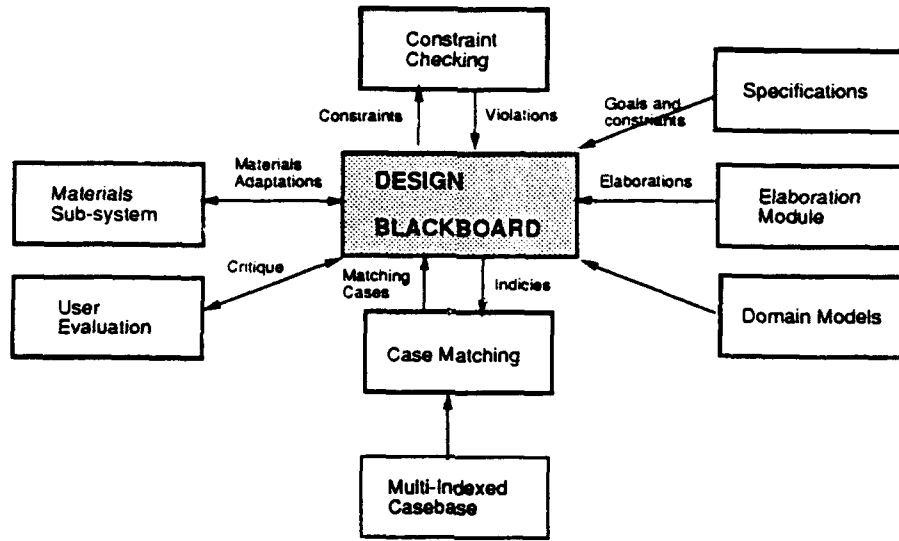


Figure 3-17: System Architecture

The specifications are input to CADET as a file that states the desired goal behavior, constraints, and physical laws. The goals are treated as indices by the Case Matching module. This module first uses a relational database to quickly find all cases that match the given index. If the design case base does not contain designs of artifacts with the same or similar function to the input case, using the given overall behavioral goal specification as an index will not retrieve any cases. Moreover, since there is no one-to-one correspondence between physically realizable device components and desired device sub-behaviors, sub-behavior matching of *pre-determined* components may not yield relevant retrievals. This problem is approached in CADET through *behavior-preserving transformation* techniques that transform an abstract description of the desired behavior of the device into a description that can be used to find relevant designs in memory [46].

The transformation is done in two ways: (a) If it is known what physical laws and principles are going to govern the solution, then the given goal is transformed by relying on the laws to achieve certain sub-behaviors. (b) If, however, the relevant laws are not known *a priori*, sub-behaviors are hypothesized and the case memory is searched to find ways in which the required behavior may be achieved. This is in contrast to other approaches that assume *a priori* knowledge of the domain laws and models that will be part of the solution [79]. If CADET cannot complete a design because it is not given the relevant physical laws, it hypothesizes behaviors and looks for cases which embody the relevant laws. Our approach has the following advantages: (1) the

system at each point in the search is aware of what behavior it is trying to achieve, (2) because cases embody design optimizations, the accessed components correspond to already optimized physical structures, (3) solutions may involve use of principles outside the given domain, that have been successful in a prior design, (4) the problem solver does not have to re-solve problem from scratch. A more detailed presentation of index transformation is given in section 5.

The result of index transformation is an *index graph* whose nodes are design variables and whose arcs are qualitative relations. Subgraphs of the index influence graph describe component sub-behaviors. Indices are used, not a set of features, but as an index graph with an associated topology. This makes case matching and retrieval a challenging task. The index graph is decomposed into separate influences. For a given index graph, all cases that match each influence are retrieved. If an influence does not retrieve any cases, a relaxed match is attempted. The relaxed matching process utilizes conceptual hierarchies of design parameters that allow decisions of parameter closeness. For example, a vertical signal is a relaxation of a horizontal signal because they are both mechanical signals. Finally, cases or snippets are selected based on criteria such as overall device cost, weight and a synthesis measure. The synthesis measure of a component case or snippet is the ease with which the snippet can be synthesized with the rest of the selected snippets in order to form a working overall device that meets the input specifications. Based on the selection criteria, the pareto optimal set is returned to the design blackboard. Section 6 presents a more detailed discussion of the matching module.

Using the index graph, CADET retrieves alternative sets of cases (and case pieces) whose synthesis gives behavior that is equivalent to the overall behavioral input specification. Design critics, such as the synthesis module and the materials module, monitor the evolving design on the blackboard. If problems are detected, then the retrieved cases must be adapted. Adaptations can range from simple parametric modifications, such as re-sizing, to those requiring sophisticated knowledge and reasoning, such as adaptation of the material of device parts. The focus of the current implementation in terms of adaptation is a materials specialist that provides material adaptation advice. The materials specialist is quite sophisticated incorporating expert material knowledge from the Ashby charts [1] and employs fuzzy reasoning. A more detailed presentation of materials adaptation is given in section 7.

3.11 CADET's Transformational Approach

Most existing case-based systems use a pre-defined set of indices to access cases in memory. This indexing strategy is limiting, since salient features of the current case that could constitute good indices may not directly match the pre-defined index set. In design, the cases have associated descriptions in terms of both behavioral and structural characteristics. If parts of the behavioral specifications of the desired design correspond to the behavioral indices of the components, then the components (and hence their structural description) can be directly accessed. However, since there is no one-to-one correspondence between the desired behavior of a device and the individual component behaviors, it is often not possible to find relevant cases by using either the given overall behavioral specification as an index into case memory, or a

pre-determined set of component indices. This gives rise to the need for a technique that can recognize underlying behavioral similarities. Our approach to this problem is based on using a combination of: (1) the behavior-preserving transformation techniques that convert an abstract description of the desired behavior of the device into a description that facilitates a similarity recognition and subsequent retrieval of relevant components, (2) matching and extracting of relevant snippets of designs in the case-base.

For example, consider the design of a device which controls the flow of water into a flush tank. The continuous part of the behavior can be specified as follows: *as the depth of water (D) in the tank increases, the rate of flow of water into the tank (Q) should decrease*. This specification may be used as a set of indices to find relevant cases in memory. If there are no cases that directly match the specifications, then it would be useful to consider using parts of several cases. In this instance, an analogically relevant case is a hot-cold water faucet shown in Figure 3-18. The faucet is specified as a device that allows for the independent control of the temperature and flow rate of water by appropriately mixing the hot and cold water streams. By extracting portions of the faucet, such as the see-saw part, it is possible to design the flush tank device as shown in Figure 2-8.

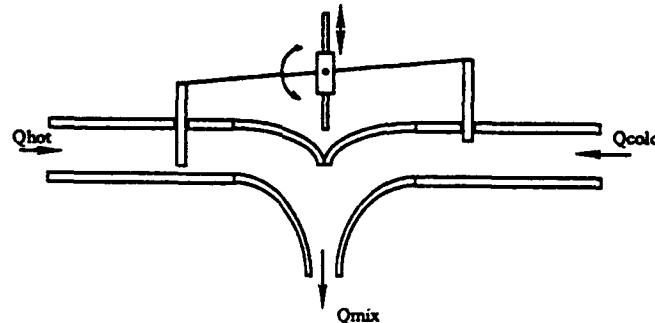


Figure 3-18: A Simple Hot-Cold Water Faucet

The relevance of the faucet case to the design of the flush tank would not have been possible to recognize because the functional descriptions of the two devices are completely different. On the surface, the base (faucet) and the target (Flush Tank) are very different. The relationship is subtle and hidden. The analogy is drawn by recognizing common behavioral patterns underlying the base and the target. Another complicating factor is that the whole faucet is not relevant to the target problem. The relevant faucet parts have to be recognized and extracted.

The design process that achieves this kind of reasoning has two parts: similarity recognition and sub-behavior matching. First, the goal specification is elaborated by applying transformation

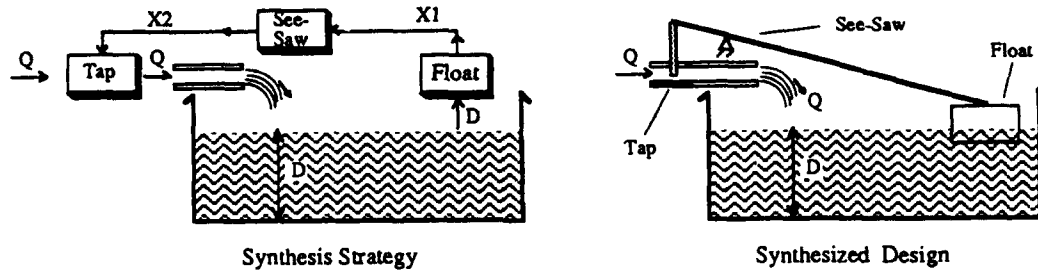


Figure 3-19: A flush tank

operators. This process, in essence, generates several alternative behavior descriptions that are equivalent to the original goal. These alternatives are then matched against the cases in memory. The matching process tries to find entire cases or parts of cases that share common "sub-behaviors" with the elaborated goal. Index transformation is a way to change the given salient features of the current cases in ways to match the indices under which previous cases have been stored, thus making accessible to the problem solver previously inaccessible cases.

The transformations do not impose any a priori structure or topology on the physical realization of the design. The decompositions are collections of sub-behaviors with information on how the sub-behaviors must interact to produce the overall device behavior. In effect, the program is insensitive to the form of the given behavior specification. Even if a given behavior specification is not compatible with the representation of the cases in memory the program is able to apply transformations to find behavior equivalence between the specification and relevant design cases in memory.

The transformation technique described here is applicable to any domain in which behavior can be modeled as a graph of influences. An influence is a qualitative differential (partial or total) *relation* between two variables one of which is a dependent variable and the other an independent variable. The notion of influences is based on the notion of confluences [6] and causality [24] in device behavior. Influences are organized as graphs. In general, an influence graph is a directed graph whose nodes represent the variables of concern and whose arcs are qualitative values depicting the influence of one variable on another. These graphs of influences are used in CADET to represent the behavior of devices, where each influence corresponds to some physical law or principle.

Consider, for example, a household water tap that has two inputs: a water source and a signal to regulate the rate of flow of water. In the two and a half dimensional representation [74] that we

have adopted, the tap is represented as a pipe with a gate as shown in Figure 2-1. The flow rate is given by Q and the position of the gate is given by X . The position of the gate controls the flow rate. This behavior is represented as an influence $Q \leftarrow^+ X$, which is read as follows: "The flow rate (Q) increases (+) monotonically with an increase in the signal (X)". This influence represents the "tap" principle.

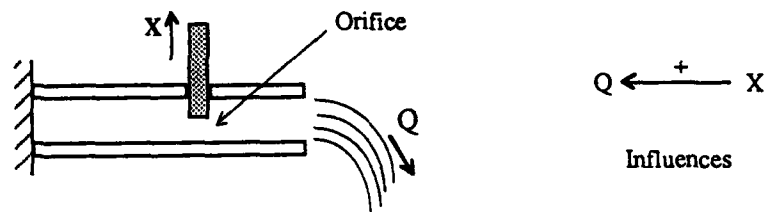


Figure 3-20: A simplified water tap

At a more detailed level, the influences correspond to standard physical laws and effects [21]. The tap's detailed behavior, as shown below, is composed of the following influences: (1) As the gate is opened, its position (X) influences the *Orifice* size and, (2) the total flow of water (Q) is influenced by the *Orifice* size and the *Pressure* difference. The first influence is based on rigid-body motion and follows from the definition of an orifice. The second set of influences are based on Bernoulli's theorem.

The transformation technique in CADET is based on two qualitative rules involving influences.

Design Rule 1. If the goal is to have x influence z , and if it is known *a priori* that u influences z , then the goal could be achieved by making x influence u .

Design Rule 2. If the goal is to have x influence z and if it is known *a priori* that some two quantities p and q influence z then, the goal could be achieved by making x influence p or q , or both.

The two design rules transform a given influence into a more detailed set of influences which are behaviorally equivalent to the original influence. As transformation operators, the first rule yields a serial transformation and the second one yields parallel transformations. The next section presents an example of how transformation integrates with case retrieval.

3.12 Matching and Retrieval of Cases and Snippets

The matching module in CADET takes an elaborated influence and uses it as an index to find relevant cases in memory. The module tries to find devices or parts of devices, whose influence graphs or subgraphs match the index influence-graph. The retrieved components are then synthesized to form a device which satisfies the specifications.

For example, assume we want to design a device that takes a high pressure signal and outputs a rotational motion. The index influence graph is given by $rotation \xleftarrow{+} pressure$. Assume that the system is unable to find cases that match this index. The next step is to elaborate the index in order to facilitate case retrieval. One possible elaboration is: $rotation \xleftarrow{+} X \xleftarrow{+} pressure$, where X is an unknown variable. The value for X is found by searching the casebase. The matching algorithm takes these two influences in the elaboration, and starts by looking for all the cases that match the first influence: $X \xleftarrow{+} pressure$. The system finds all the cases where pressure, affects some unknown. Let's suppose that the system finds a tank with a pipe ($waterflow \xleftarrow{+} pressure$) and a pressure cooker ($boilingtemperature \xleftarrow{+} pressure$). These cases represent two possible matchings for the first influence. The system then proceeds to the second influence. For the tank, the unknown variable X is bound to the *water-flow*, and so, the second influence becomes: $rotation \xleftarrow{+} water-flow$. This influence retrieves a water wheel from the casebase. The pressure causes the water to flow, and the water causes the water wheel to turn, producing a rotation as output (figure 3-21). On the other branch of the search, the system fails to find a case that produces rotation from a change in boiling temperature. In this way, the matching subsystem takes a given graph of influences and finds cases that match the index.

Matching subgraphs of an index with graphs and subgraphs of cases in the casebase, is a variant of the subgraph isomorphism problem. We approach the problem by separating the graphs into individual influences. Each influence is of the type: (*A influences B positively/negatively*). Influences of each case are indexed separately in a relational database. For a given index graph of influences, the system finds all cases that match each influence.

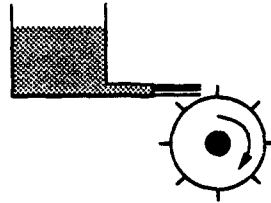


Figure 3-21: A Turbine

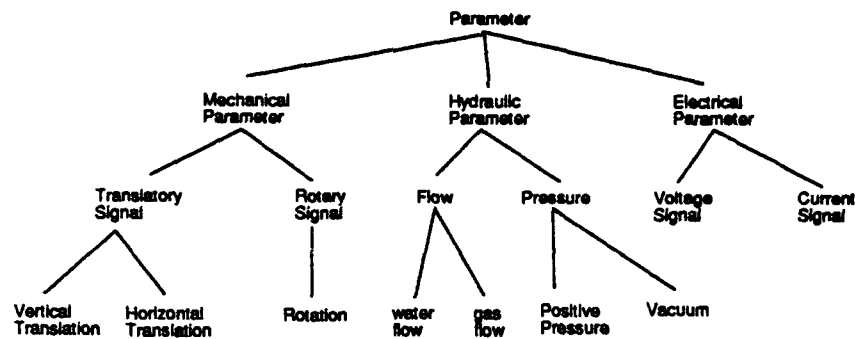


Figure 3-22: The Parameter Hierarchy

If an index influence has an unknown variable, then the system will find all possible bindings for that variable. Each possible binding is treated as a new alternative. Each binding is propagated to the remaining influences in the index. If there are still some unknowns left in the index, all possible bindings of those variables are also treated as new alternatives. The subgraph search grows as a tree, wherein, at each level we have one or more alternatives to choose from. To control the search, the A^* algorithm is used to select the most promising paths. In CADET we use several criteria to guide this search: total cost, weight, and a synthesis measure. The synthesis measure of a component or a device is the ease with which the component or device can be synthesized with a given set of components to form a working device. The measure helps

prune away awkward designs that are difficult to adapt. The synthesis measure is based on the following factors:

- It has been observed that selecting components of the same device type for as many index influences as possible results in a more easily synthesizable solution. If we select a device or a component of the same device type as it's parents it can be easily assembled with the other components. Hence a component which is of same device type as it's parent is given higher priority. For example, water flow is consequently closer to pressure than to a rotating motion.
- The distance between the influence parameters is another part of the synthesis measure. The distance between any two parameters is measured as the number of levels one has to go upwards in order to find the parent node that is common to both parameters. This factor has a major contribution to the synthesis measure. Consider the following index influence graph:

rotational-signal $\xleftarrow{+}$ unknown-sig1 $\xleftarrow{+}$ translatory-signal

Assume, that the first influence has two matching cases in the case-base:

- (a) flow-signal $\xleftarrow{+}$ translatory-signal (case: simple slider tap)
- (b) rotational-signal $\xleftarrow{+}$ translatory-signal (case: rack & pinion)

Depending on the choice for the first influence we have the following alternatives for the second influence:

- (c) rotational-signal $\xleftarrow{+}$ flow-signal (case: water wheel)
- (d) rotational-signal $\xleftarrow{+}$ rotational-signal (case: gear pair)

The second alternative is easier to synthesize because, both the gear and the rack & pinion are mechanical devices. The other option (tap and water wheel) that involves mechanical and hydraulic action, is harder to synthesize.

The case retrieval algorithm is as follows:

-
1. Create two empty lists: LIST and RESULTS.
 2. Read the given index influence-graph. Put it in LIST.
 3. If LIST is empty, then return the RESULTS and STOP.
 4. Remove the first graph in LIST. Call it GRAPH.
 5. If all influences have been previously marked as "matched"

Then put GRAPH in the list called RESULTS. Go to Step 3.

Else, Select (don't remove) the first unmatched influence in GRAPH. Call it INDEX.

6. Search the database for all influences that match INDEX. For each match, also retrieve the corresponding case name.

If INDEX has no unknowns, associate the names of all the matched cases with INDEX. Mark INDEX as "matched". Add the GRAPH back to LIST. Go to Step 6.

If the influence has an unknown variable, find all influences in the casebase that match the known part of the influence. Call these influences MATCHINGS.

- a. For each influence (X) in MATCHINGS, create a copy of GRAPH, replacing INDEX by X. In each copy of GRAPH, replace all occurrences of the unknown variable in INDEX, by the corresponding variable in X.
 - b. Mark each X to indicate that it has already been matched. Also, associate with each X the names of all the cases that contain the influence.
 - c. Add the copies of GRAPH to LIST.
- † the LIST, based on pareto optimal satisfaction of all the given criteria (e.g. reliability, and materials). This step is optional.

8. Go to Step 3.

Each graph in RESULT is finally checked for synthesis complexity. Case alternatives for each influence are treated as graph colors. A clique partitioning algorithm is run on each graph to find the cases that will be the best to synthesize.

3.13 Material Adaptation

Adaptation is an important process in design for several reasons. First, since the specifications of the current problem do not often exactly coincide with the specifications of previous problems, adaptation is necessary to fit retrieved relevant cases to the current specifications. This reason for adaptation is the one most commonly found in CBR systems. Second, design is an underspecified task where goals and constraints are dynamically generated during the design process. Generation of new goals and constraints is the result of (1) the case based reasoning process itself, where the new goals or constraints are present in retrieved cases, and/or (2) creativity on the part of the designer. Adaptation is necessary to incorporate the new goals and constraints into the evolving design [12, 20]. Third, since new designs are synthesized from component parts, interactions arise during the synthesis process. These interactions often result in side effects that alter the device desired behavior. Hence, adaptation is necessary to take care of the side effects.

The current focus in CADET is adaptation due to new goals, constraints and side effects that concern materials considerations. Material adaptation involves re-evaluating the reasons why a particular material was used in a precedent case and whether the same or new criteria (goals and constraints) are relevant in the current problem. Once the new criteria have been established, appropriate materials are selected and substituted. For example, if a household oven were being adapted for an aerospace application, the designer would have to re-evaluate the material choices

in terms of weight minimization. This evaluation could result in replacing heavy steel components by ceramic-coated aluminum ones.

Adaptation of a component of a retrieved precedent from the point of view of materials, may take the following forms: (1) increase or decrease in the loading level (e.g., higher loads may require a stronger material), (2) change in mode of loading (e.g., torsion instead of bending alone), (3) addition or removal of constraints (e.g., the material of pipes used for drinking water should be non-toxic), and (4) goal changes (e.g., weight considerations become very important in aerospace applications).

We will now examine the material selection and substitution subsystem in CADET. The ideas are presented through an illustrative example. In our example, we will see how a base case (a see-saw) is re-used in a target problem (a faucet). Material justifications in the base are modified to suit the target context. This leads to the selection of new materials.

3.13.1 Case Representation

Consider the simple see-saw shown in Figure 3-23. The see-saw's main component is a beam, hinged at its center. Material selection for the beam is based on a variety of criteria: as it is intended for children it should not be too heavy; as it is intended for outdoor use it should be water resistant; it should not rust; should not cost too much; and it's color should not fade in the sunlight.

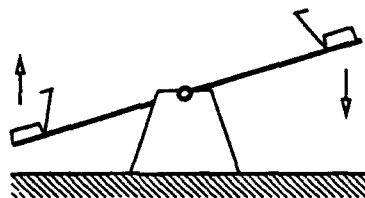


Figure 3-23: See-saw on a playground

The see-saw case is represented in the database as a frame. The material justifications are provided in terms of physical properties.

See-saw

Location: playground

Material: wood-parallel-to-grain

INHERITED SPECIFICATIONS

Mode of loading: bending of rods and tubes

Normal load: support a weight of 100 kg

BEHAVIORAL JUSTIFICATIONS

Bending resistance: high (or better)

reason: mechanical "to avoid bending under normal load"

Torsion resistance: medium (or better)

reason: mechanical "to avoid torsional failure"

Fracture resistance: high (or better)

reason: mechanical-safety "to avoid sudden failure"

FUNCTIONAL JUSTIFICATIONS

UV radiation resistance: at least good

reason: aesthetic "to avoid color fading in sunlight"

Temperature of use: very cold TO very hot

reason: safety "avoid skin contact
problems at extreme temperatures"

There are two types of justifications used in the system: behavioral and functional. A reason is attached to each justification. A degree of importance can also be added to the justifications that would convey the extent to which the justification must be taken into account in determining the right material class. For instance, one may consider aesthetic reasons less important and therefore focus first on satisfying mechanical and safety constraints.

The *behavioral justifications* constitute the intrinsic reasons that led to the material choice. A behavioral characteristic of a see-saw found on a playground is to support the weight of two people sitting at both ends. The mode of loading of this artifact pertains to the mode of loading: "bending of rods and tubes". The mode of loading determines the relevant mechanical properties the material should possess. In the see-saw case, bending resistance, torsion resistance and fracture resistance are the three important properties.

The *functional justifications* correspond to a particular instance of the artifact. Let us say, we are talking about an outdoor location for the see saw. If the see-saw is intended to be used in very cold weather, one would want to avoid metals because of possible skin contact. The other functional justification relates to an aesthetic point of view. One doesn't want the color of the see-saw to fade too quickly when exposed to the sun's radiation.

3.13.2 Material Adaptation Process

Now consider the target application for the see-saw: a water tap. Figure 3-24 depicts a tap using the see-saw principle. The rotation in the vertical plane regulates the relative quantity of hot and cold water. The vertical motion of its hinge increases or decreases the combined flow. The tap represents a target application where the environmental conditions are quite different from that of the see-saw.

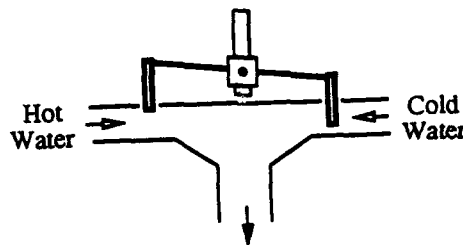


Figure 3-24: Tap using the see-saw concept

The process of finding the right material for the water tap involves the following steps:

1. Look up the material and material justifications of the base case.
2. Look up the features of the target artifact.
3. Deduce (possibly new) material properties the base component should possess in the target context.
4. Infer the values (fuzzy) that these properties should have.
5. Suggest the best materials for substitution.

In the tap example, the mode of loading remains the same, as well as the three important mechanical properties. However, the forces on the see-saw mechanism is much lower. The designer recognizes this and inputs the following to the system:

bending resistance: medium (or better)
 torsion resistance: medium (or better)
 fracture resistance: low (or better)

Orders of magnitude may be used here rather than actual values. This facility allows designers to work with incomplete or inexact information about an evolving design concept. These quantities can easily be represented as fuzzy intervals [7]. We have divided the whole range of possible values in five categories: *very low*, *low*, *medium*, *high* and *very high*.

Functional justifications can also be modified. In the water tap, resistance to ultra-violet radiation is not required, and temperatures below freezing are unlikely to be encountered. Therefore, the corresponding justifications attached to the retrieved case can be discarded. In addition, one may require the artifact to be resistant to salt corrosion if, for instance, we are dealing with a marine environment. Having modified the criteria retrieved by the system, the designer can now use the system to find relevant materials. The selection step is done using special material selection charts that were developed by Prof. Ashby of Cambridge University [2]. Our program is able to use fuzzy specifications to locate appropriate materials in the charts.

The Ashby charts show the relative and absolute positions of various material classes with respect to a tradeoff between two different properties. For example the sketch in Figure 3-25 presents the positions of the materials classes A, B, C, D based on their strength and density.

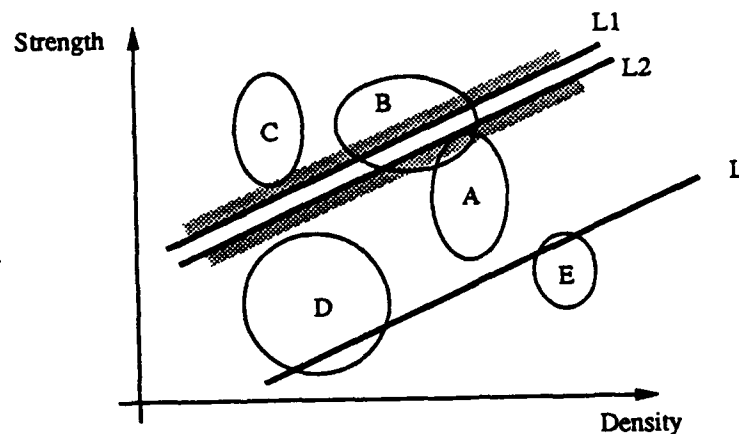


Figure 3-25: Sketch of a materials selection chart.

The oblique line L crosses material classes with the same strength to density tradeoff. The higher this line on the chart the better the ratio. Looking for materials with a particular combined property (tradeoff between two properties) is a problem that involves two kinds of imprecision. First, the material classes plotted on the charts represent the smallest area containing all the materials of some class. This means that within an envelope there may be some regions that do not correspond to any actual material. Secondly, one rarely looks for materials with a precise value of the combined property but rather for materials within acceptable ranges of values.

This notion is exemplified in Figure 3-25. A designer may look for materials that lie between two oblique lines L_1 and L_2 . The area between L_1 and L_2 is the preferred range of values for the combined property. The ranges of values next to lines L_1 and L_2 (shown by the shaded area on Figure 3-25) represent values close to the preferred range. Materials in these areas should not be totally rejected although they do not pertain to the set of the preferred materials. This defines three conceptually different regions: (1) above the upper shaded area or under the lower shaded area are the materials one is definitely not interested in, (2) materials between L_1 and L_2 definitely meet the required combined property, and (3) the materials crossed by one of the shaded areas but outside the interval $[L_1, L_2]$ are acceptable to a certain extent (they are not totally rejected). Actually, in this third region, the closer the material to the stripe $[L_1, L_2]$ (the preferred threshold) the better.

This type of imprecision can easily be handled by making use of fuzzy set theory. This theory enables the ranking of material classes with respect to their degree of preference, given some property. For example one could say that material class B meets the requirements more adequately than material class A and much more than materials class C . Also, the fuzzy set theory provides a means to aggregate several material rankings relative to different requirements.

3.14 Conclusion

We have presented a system that aids in the conceptual design of mechanical devices using a case base of previous designs. The process consists of applying behavior-preserving transformations to an abstract description of the desired behavior of the device until a description is found that closely corresponds to some collection of relevant cases.

The approach allows retrieval of analogically related cases from other domains. If CADET does not find an exact match it relaxes its requirements, looking for close matches. This sometimes leads to the retrieval of too many relevant cases. This problem is addressed by the use of a multi-objective A* type algorithm for the selection of cases. The selection is based on criteria that prefer alternatives that exhibit design simplicity, ease of synthesis, and low cost. As a result of evaluation of retrieved cases, adaptation with respect to materials is also performed.

Our approach has the following advantages: (1) the system at each point in the search is aware of what behavior it is trying to achieve, (2) because cases embody design optimizations, the accessed components correspond to already optimized physical structures, (3) solutions may involve use of principles from outside the given domain that have been successful in a prior design, (4) the problem solver does not have to re-solve problems from scratch.

Appendix: A look at the System

In this section, we present a partial trace of the CADET system as it aids the designer in synthesizing two hydro-mechanical devices.

The first example is the design of a hot-cold water faucet. The behavior of the faucet is described in terms of qualitative relations. The input file to cadet is shown below:

```
;;
;; This input file describes the specifications
;; for the design of a household faucet.
;; The input is described as a set of frames.

;;THE TASK:
;; Design a device that mixes hot and cold water,
;; and allows for independent control of temperature and flow rat
;; of the mix. This will be done by two signals, the temperature
;; control signal and the flow control signal, that
;; are externally input.

;; Setting up the design parameters:
(cschema 'tc ('is-a 'temperature)) ; tc = temp of cold water
(cschema 'th ('is-a 'temperature)) ; th = temp of hot water
(cschema 'tm ('is-a 'temperature)) ; tm = temp of mix
(cschema 'qc ('is-a 'flow)) ; qc = flowrate of cold wat
(cschema 'qh ('is-a 'flow)) ; qh = flowrate of hot wate
(cschema 'qm ('is-a 'flow)) ; qm = flowrate of mix
(cschema 'st ('is-a
    'vertical-displacement)) ; st = temp control signal
(cschema 'sf ('is-a
    'vertical-displacement)) ; sf = flowrate control sig

;; Task Description:
(cschema 'faucet
;; Goals: Temperature-of-mix (Tm) increases with Temp-Signal (St)
;; Flow-of-mix (Qm) increases with Flow-Signal (Sf)
('goals '((tm st +) (qm sf +)))

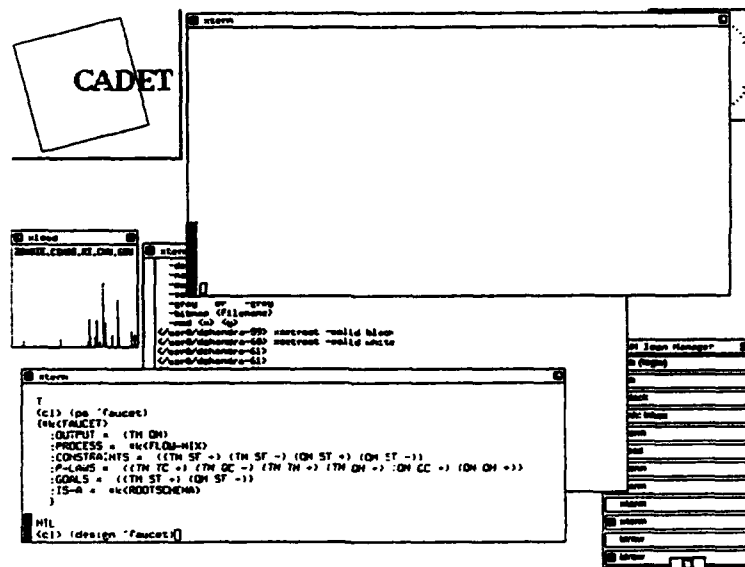
;; Conservation of Mass and Energy
;; For example, (tm qh +) means, as the quantity of the hot water
;; increases, the temperature of the mix (tm) also increases (+).
('p-laws '((tm tc +) (tm qc -) (tm th +) (tm qh +)
    (qm qc +) (qm qh +)))

;; Constraints are specified as influences that are invalid.
;; Flow signal should not influence the temperature of the mix
;; Temperature signal should not influence the flow rate
('constraints '((tm sf +) (tm sf -) (qm st +) (qm st -)))
```

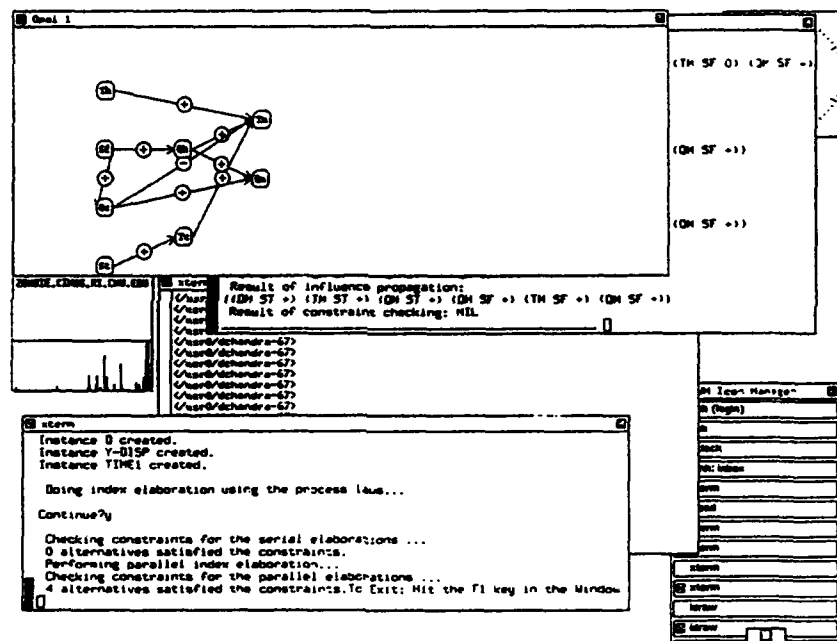
```
( 'process      'flow-mix) ;; process type
( 'output      '(tm qm)) ;; outputs of device
```

The variables that depict temperatures and flowrates are defined in terms of primitive concepts that CADET knows about. The program maintains an internal hierarchy of the various variable types. This hierarchy is used for *similarity matching*.

Screen 1: The synthesis process starts by giving the command: `(design 'faucet)`. CADET starts by looking for cases that directly match the given behavior description. If cases are not found, index elaboration begins.

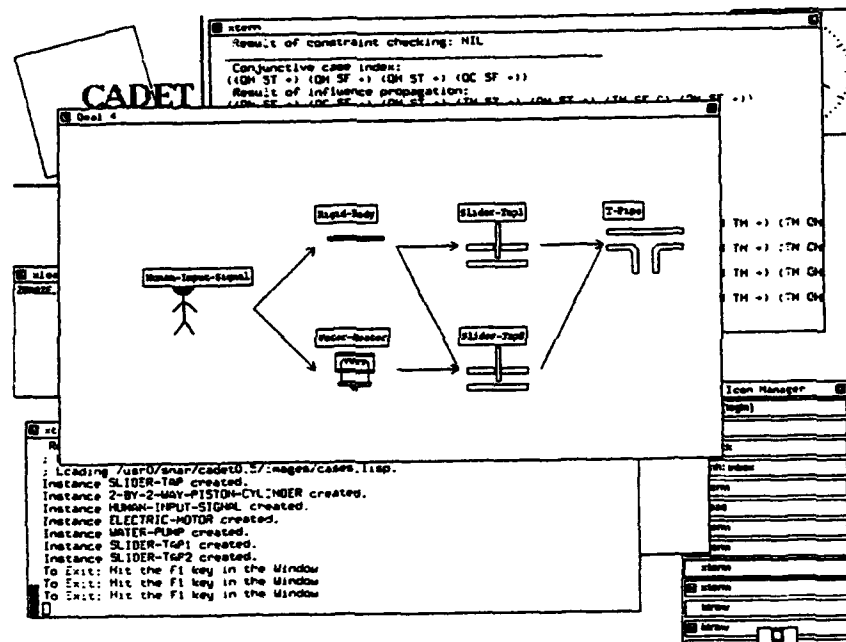


Screen 2: When the program starts its process of index elaboration and constraint checking, it displays the elaborations as it goes along. In the elaboration shown below, the temperature signal St directly influences the temperature of the cold water Tc . The program is suggesting that we control the temperature of the mix Tm by directly controlling the temperature of the cold water stream. The flow signal Sf controls the two flow rates Qc and Qh . By varying the flow signal, the total amount of water going through this device can be controlled.



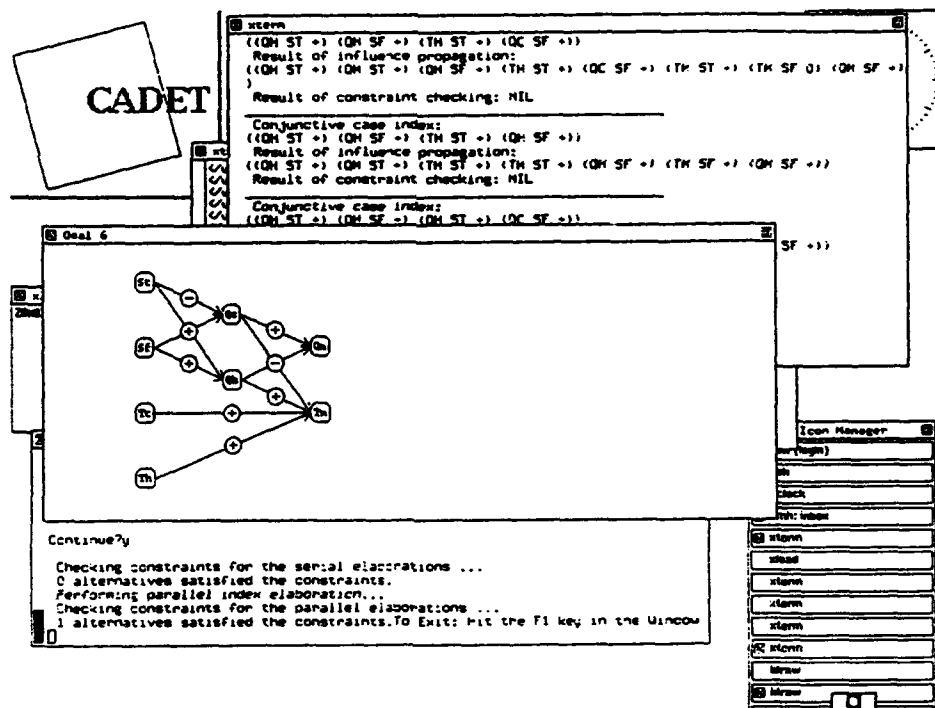
Screen 3: After the elaborations are over, the program tries to find relevant cases in the case memory. As seen below, CADET finds that a water heater will allow a given signal to positively control the temperature of a water stream.

The primary input signal comes from a human (one of the cases the program has access to). The temperature control signal is used to control the water heater, and the flow control signal operates two taps connected by a rigid body. Finally, the hot and cold water flows merge in a T-pipe.

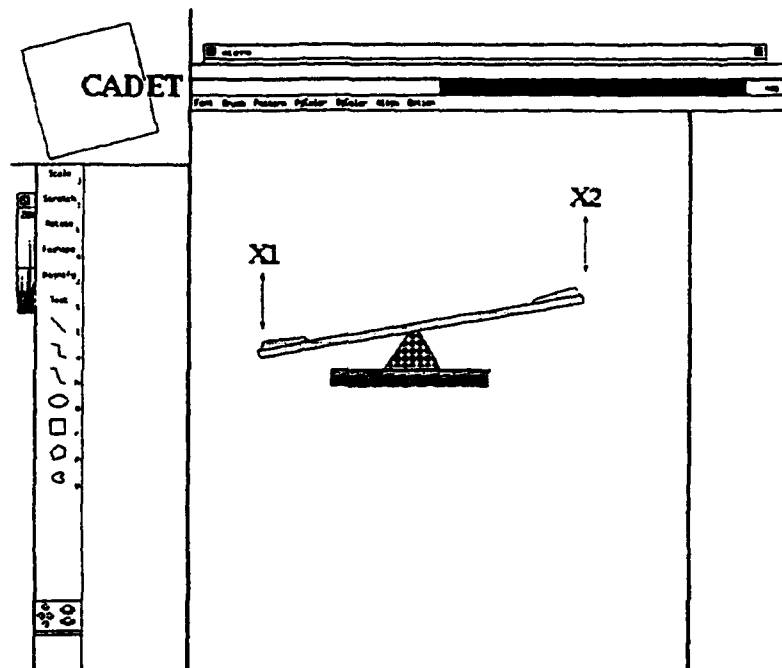


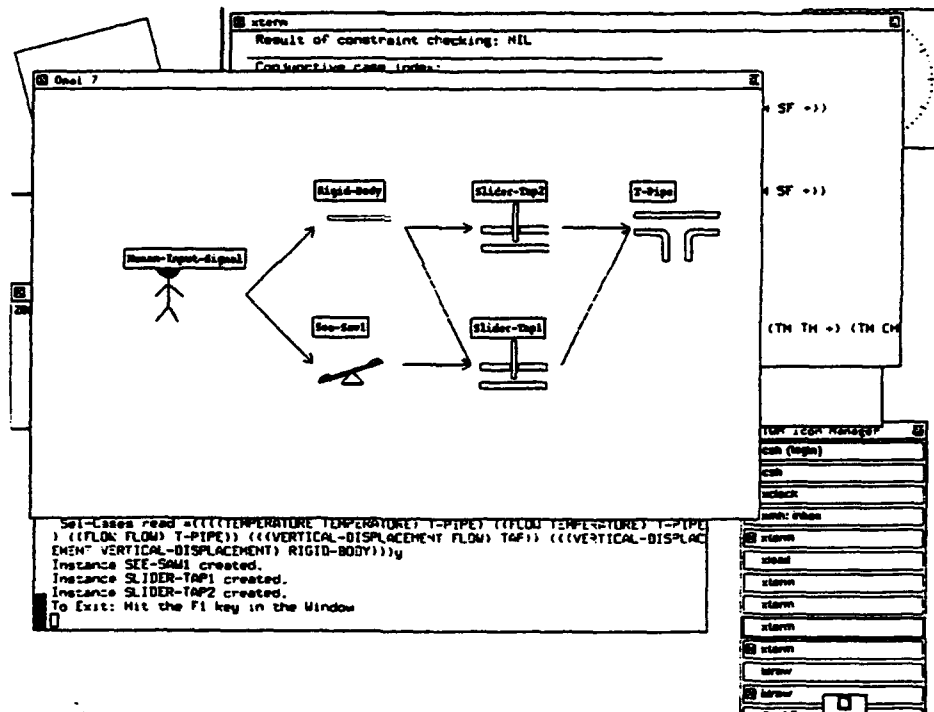
Screen 4: CADET has given us a plausible design for a hot-cold water faucet. The design, however, is not what we intended. We had not intended to use an external source of energy, such as a water heater. A new set of constraints is now added to the list of constraints in the input file. The new constraints are: (th st +) (th st -) (tc st +) (tc st -). This states that the temperature signal (St) should not influence (either positively or negatively) the temperature of the hot water (th). The same holds for the cold water stream.

CADET is invoked with the modified constraints to yield the following new elaboration (below). Note that the temperature and flowrate control signals do not effect the temperatures. The system has discovered a way of using mixtures of the two streams of water to achieve its goal. Refer back to the original specification file. There is no hint of performing temperature control by mixing hot and cold water, and the system does not have any rules either. This discovery is purely the product of elaboration.

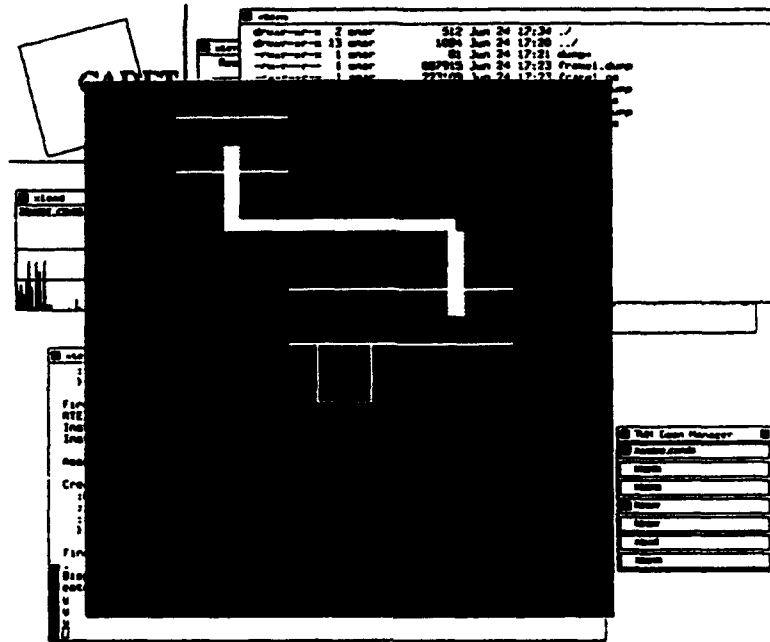


Screen 5: Once a satisfactory influence graph is found, case retrieval is re-attempted. Using the elaboration shown in Screen 4, CADET finds a see-saw as a suitable way of controlling temperature by mixing hot and cold water streams.

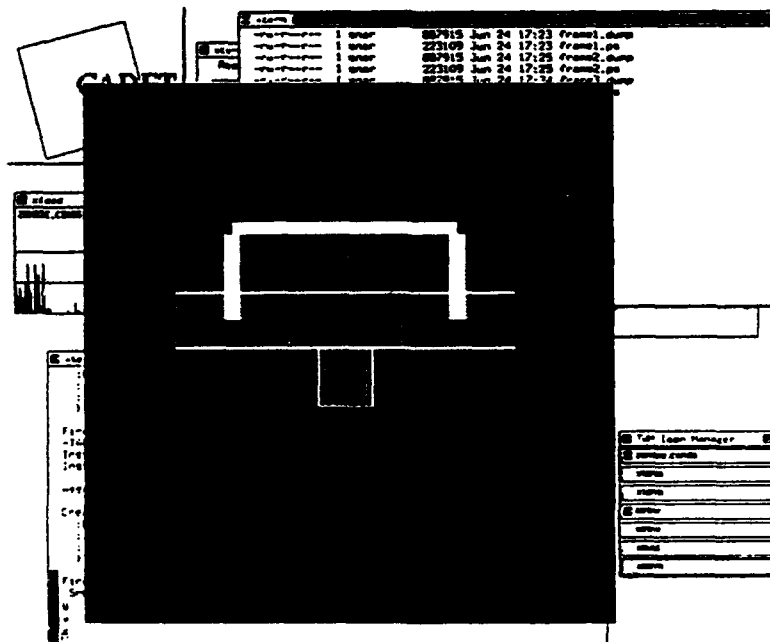




Screen 7: After adaptation (See Section 6.4), CADET tries to physically synthesize the cases it has retrieved. This process is done using a vector matching approach. The attempt shown below is a failed case, because this arrangement of the snippets will not behave correctly. We are currently relying on the user to reject wrong configurations.



Screen 8: At the fourth (final) attempt, CADET generates the right vector alignments to produce a workable design. After the designer accepts a design, the casebase is updated with the new design. The new case is available for future problem solving.



3.14.1 A Metal Bending Device

This section shows CADET designing a metal bending device. Of the various devices that CADET has designed, this example shows how cadet re-uses a faucet device that it had designed earlier.

As before, the input specification file is typed in manually:

```
;; Design a device that creates up-and-down
;; displacement by elastic-deformation

(cschema 'metal-bender
  ;; As time increases the deformation is to
  ;; increase and decrease. L = { -, 0, + }
  ('goals '((d time1 L)))

  ;; Deformation is proportional to a vertical (y) displacement
  ('p-laws '((d y-disp +)))

  ('constraints)
  ('process 'elastic-deformation)

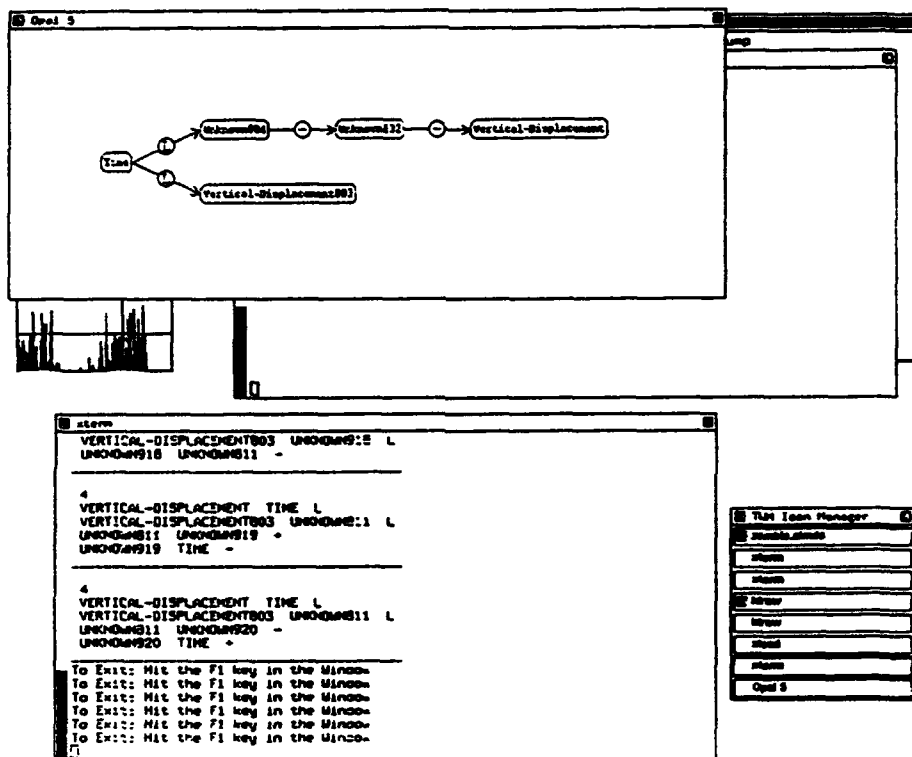
  ('output '(d)))

(cschema 'd ('is-a 'deformation))

(cschema 'y-disp ('is-a 'vertical-displacement))

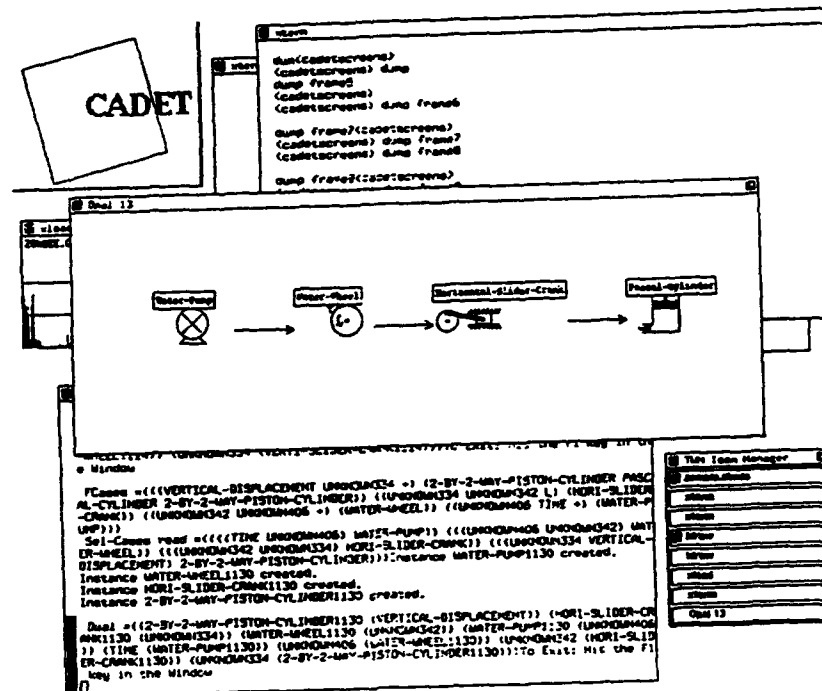
(cschema 'time1 ('is-a 'time))
```

Screen 9: CADET starts with index elaborations. Because there are no given physical-laws, the program elaborates using unknowns. The screendump below shows CADET considering an elaboration with two unknown quantities. After the program has done a few levels of elaborations, it makes a call to the casebase manager and finds relevant matches.



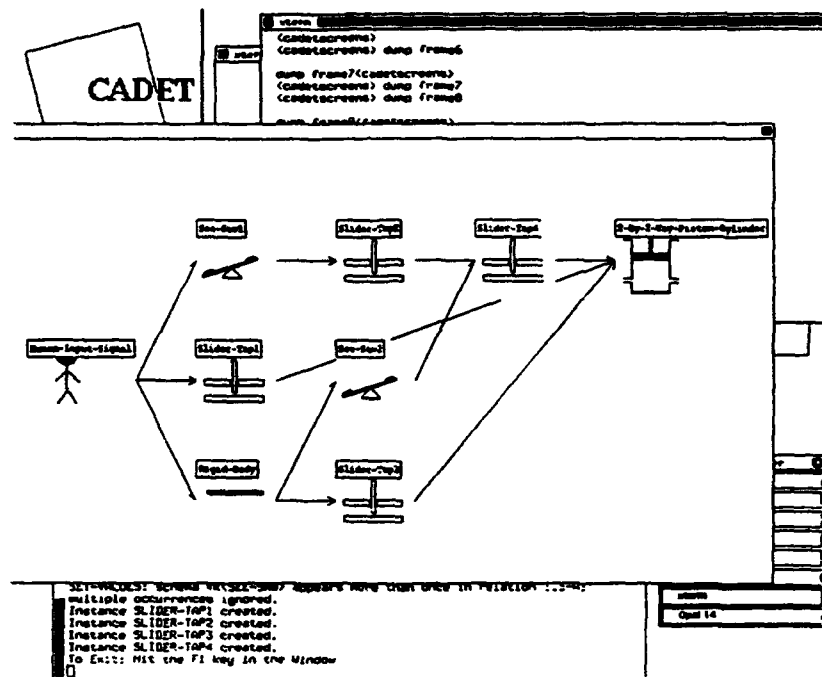
Screen 10: Below, is one of the alternatives that CADET finds. This example is a result of CADET's serial influence elaborations.

The program is suggesting using a Pascal Cylinder that is driven by a slider crank mechanism. The crank is attached to a water wheel that is run by water from a water pump. The user rejects this alternative.

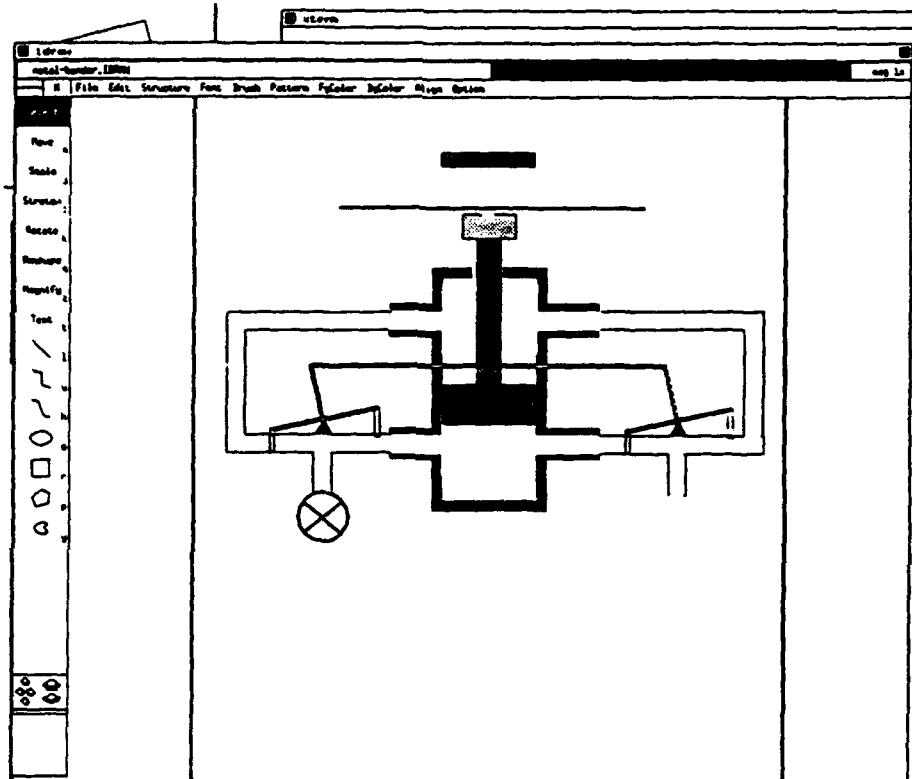


Screen 11: The following screendump shows a design alternative generated as a result of parallel elaborations. A Piston Cylinder is used to create a strong upward motion. The cylinder is driven by a combination of taps that are used to fill and exhaust the cylinder. The user accepts this design.

The program is suggesting the use of a human input signal to control the inflow and outflow from the cylinder. The combination of taps and see-saws is taken from the faucet case, that the system just designed. Every new problem solving process ends with storing the current case into memory.



Screen 12: The final design (note that it uses a faucet) is shown below.



Chapter 4

Reasoning about Connections in Synthesis

Common design practices rely heavily on searching and studying of prior designs, patents, past design rationale, standards and new product announcements. An important step toward automating these activities is the creation of repositories of design information indexed by abstract attributes in addition to low-level structural descriptions. By reusing prior designs or their components an engineer can save design time, by leveraging off previous worked-out solutions, and avoid repeating past mistakes, by accessing information on manufacturing or field failures linked to the retrieved design. Under the case-based design paradigm, an engineer combines parts of different design cases to synthesize a device that satisfies a useful need. Physical synthesis entails taking into account possible interactions between components and sub-assemblies as well as reasoning about the dynamics of the system. In this paper we present a methodology for physical synthesis of design cases and components retrieved by a case-based design tool. Connecting elements for the design cases and components are retrieved from a casebase of connections. Indexing of these connections is based on the mobility restrictions that they impose on the connected parts. The information necessary to accomplish this task is still of a high-level nature, namely, the topology of the artifact and its abstract behavior specification.

4.15 Introduction

Global competition, rapid advances in technology, and an increased demand for high performance have a major impact on the way products are designed and manufactured today. To remain viable, an organization must reduce design cycle time, improve design quality, avoid major design errors, and increase its ability to respond to change. An important step toward meeting these requirements will be the development and proliferation of design repositories that provide easy access to information about previous designs, new products, technological breakthroughs, and past failures. By reusing prior designs an engineer can save design time by leveraging off previous worked-out solutions. Moreover, the designer can avoid repeating past mistakes by accessing information on manufacturing or field failures linked to the retrieved design. This information can also be used to update or adapt an old design in response to changes in technology, government regulations or market preferences.

The connection between case-based reasoning (CBR) [30] and traditional engineering design is clear and well studied [46, 47, 67, 65, 71]. CBR is the method of using previous cases (problem solving experiences) in the solution of a new problem. Given a new problem, appropriate previous cases are retrieved from a database, and differences as well as similarities between current and previous cases are identified. These similarities and differences are used to select and adapt the retrieved cases to fit current circumstances. At the end of problem solving, the solved case is stored as a new case to be used in the future. Similarly, engineering designers often combine parts of different design cases to devise an artifact that satisfies a useful need [59, 75, 76]. The parts of an artifact are often highly-integrated and tightly-coupled to render the required function. Although there is no simple and clear correspondence between the functional characteristics of a device and those of its components, the overall behavior⁸ of the device can be decomposed into sub-behaviors of its components⁹, as shown by Sycara and Navin Chandra in their work on case-based design [46]. They use a graph-based representation of behavior and apply behavior-preserving transformations to an abstract description of the desired behavior until a description is found that closely corresponds to some collection of relevant cases. This approach allows for retrieval of cases without imposing a predetermined decomposition of the design and it also utilizes knowledge of domain laws. Innovative designs result when the design parts or components are taken from design cases that are functionally dissimilar to the current design problem.

Three types of design problems or stages for the design process are recognizable [9]: conceptual, configuration and parametric design. In conceptual design, a functional requirement is transformed into an idealized physical concept termed embodiment or configuration (e.g., a beam). In configuration design, a physical concept is transformed into a configuration with a defined set of attributes (e.g., an I-beam). In parametric design, values are assigned to the attributes of the configuration. During the conceptual and configuration stages, design constraints are generated which substantially affect design decisions taken later in the design. In general, computer-based systems to assist in designing have been specific to each class of design problem. A case-based design system, however, allows for retrieval of cases with well-defined design attributes. Therefore, case-based design may allow for simultaneous conceptual and configuration design, hence providing cohesiveness to the design process.

The aforementioned research on case-based design has mainly focused on such aspects as behavior representation, and indexing and retrieving of design cases at a high level of abstraction. Assembly of the design components and sub-assemblies at a more detailed level were left to the human designer who had to adapt the geometry of the retrieved design cases and

⁸A device's *behavior* is what it does, while its *function* is what it is used for. For example, a clock has the behavior of moving its hands, while it has the function of telling time. The same function can be achieved through different behaviors. For example, digital and analog clocks tell time.

⁹This is achieved by transforming a given behavior specification into alternative decomposable forms that preserve the overall desired behavior

chose the appropriate connecting elements, so the resulting design perform as specified. Automation of this phase of case-based design, where component interactions are recognized and resolved, is an essential step to have a fully functional case-based design system.

The aim of this paper is to present a methodology for physical synthesis of design cases and components retrieved by a case-based design tool. The principles and synthesis strategy to be presented have been implemented to assist a designer during the configuration design phase and to provide a case-based design system with configuration design capabilities. The information necessary to accomplish this task is still of a high-level nature, namely, the topology of the artifact and its abstract behavior specification. In fact, constraints on shapes and dimensional parameters could be an end result of this stage of the design.

Connecting elements for the design cases and components are retrieved from a casebase of connections. Indexing of these connections is based on the mobility constraints that they impose on the connected parts. That is, mobility constraints (i.e., restricted degrees of freedom) compatible with the overall behavior of the device are used as indexes for the retrieval of connections and subsequent physical synthesis of components and sub-assemblies. As mentioned above, an end result of this process is the generation of constraints on dimensional parameters which have to be taken into account during detailed design. The rest of the paper is organized as follows. The next section provides an overview of a case-based design system, CADET. We then present the case-based design process and propose a typology for connecting elements used in physical synthesis of devices. This is followed by a presentation of connectivity conditions to be observed by a viable design. Then an example of physical synthesis is presented. In the appendix we briefly describe the general mobility criterion, and the effect of dimensional and geometric constraints on the realization of a design.

4.16 A Case-Based Design Tool

Sycara, Navin Chandra and Narasimhan have developed the CADET (CAsE-based DEsign Tool) system, one of the first tools for automated synthesis of mechanical designs that successfully exploits prior designs for reuse in other contexts. CADET takes as input a high level behavior specification of a desired artifact and retrieves prior designs or design parts whose composition delivers the required behavior. CADET's behavior-based indexing algorithms, composition schemes, and reuse methods have been extensively reported and referenced in the literature [46, 47, 64, 67, 66, 65, 69, 70, 71].

In CADET, the behavior of a device is represented as a collection of influences between the various behavior variables¹⁰. We propose in table 4-1 a taxonomy for the behavior variables present in behavior influence graphs (BIG's) for different energy domains. This taxonomy allows the mapping of behavior influence graphs and bond graphs [25, 49, 54] with the

¹⁰Behavior variables are those inputs, outputs and state variables chosen to characterize a system's behavior at a high level of abstraction.

subsequent advantages the latter possesses for dynamic-system simulation. An influence is a qualitative differential relation between two variables, one of which is considered the independent variable and the other the dependent variable. Behavior influences are based on the notion of confluences [6] and causality [24]. Causality assignments determine the dependency and independence of the behavior variables hence the direction of the corresponding influences. In general, a behavior influence graph is a directed graph whose nodes represent behavior variables and whose edges have assigned qualitative values depicting the influence of one variables on another. In the following, we briefly review and illustrate the process of index elaboration and matching. For more details of the approach see [46].

Table 4-1: Taxonomy of behavior variables

1. FLUIDIC FLOW_VOLUME (QH) FLOW_RATE (FH) FLOW_RATE-LIQUID (FH) FLOW_RATE-LIQUID-WATER (FH) FLOW_RATE-LIQUID-OIL (FH) FLOW_RATE-GAS (FH) FLOW_RATE-GAS-AIR (FH) PRESSURE (EH) PRESSURE-LIQUID (EH) PRESSURE-GAS (EH) PRESSURE_MOMENTUM (PH)	5. FORM LENGTH AREA VOLUME ANGLE MASS MOMENT_OF_INERTIA
2. KINEMATIC TRANSLATION TRANSLATION-DISPLACEMENT (QT) TRANSLATION-VELOCITY (FT) TRANSLATION-ACCELERATION ROTATION ROTATION-DISPLACEMENT (QR) ROTATION-VELOCITY (FR) ROTATION-ACCELERATION	6. ELECTRIC_MAGNETIC VOLTAGE (EE) CURRENT (FE) CHARGE (QE) FLUX_LINKAGE_VARIABLE (PE) ELECTRIC_ENERGY MAGNETIC_ENERGY ELECTRIC_POWER MAGNETIC_FLUX_DENSITY CAPACITANCE RESISTANCE INDUCTANCE
3. KINETIC FORCE (ET) LINEAR_MOMENTUM (PT) MOMENT (ER) ANGULAR_MOMENTUM (PR) STRESS	7. MECHANICAL_ENERGY GRAVITATIONAL_ENERGY KINETIC_ENERGY ELASTIC_ENERGY MECHANICAL_POWER
4. THERMAL TEMPERATURE (Et) ENTROPY_FLOW (Ft) HEAT_ENERGY HEAT_FLUX	8. TEMPORAL TIME FREQUENCY PERIOD

FIRST LETTER

E: Effort
 P: Momentum = $\int E dt$
 F: Flow
 Q: Displacement = $\int F dt$

SECOND LETTER

T: Mechanical Translational Systems
 E: Electrical Systems
 R: Mechanical Rotational Systems
 H: Hydraulic Systems
 t: Thermal Systems

Consider the design of a device that controls the flow of water into a flush tank. The behavior of the flush tank can be specified as follows: *As the water level (D) in the tank increases, the rate of water flow into the tank (Q) should decrease.* This behavior specification is expressed in terms

of the influence $D \rightarrow Q$, and this influence is used as an index for retrieving cases. If no case can be found that matches a particular design specification, it will be desirable to find cases whose composed behavior is equivalent to the specified behavior.

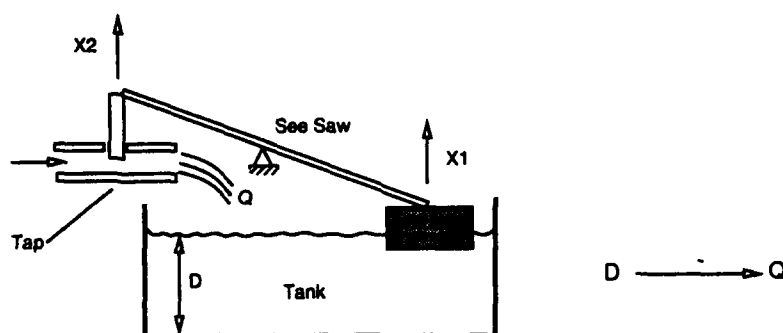


Figure 4-26: A flush tank and its behavior specification

Figure 2-8 shows a possible solution to the problem. Three cases have been identified (a float, a seesaw, and a tap) and correspondingly represented by the following influences: $D \xrightarrow{+} X1$, $X1 \xrightarrow{-} X2$ and $X2 \xrightarrow{+} Q$. The combined effect of these influences is equivalent to that in the original specification. To find cases whose composition will deliver the specified behavior, the system first automatically decomposes the specified behavior into a behaviorally equivalent set of influences and then uses each influence as an index for matching precedents in the casebase.

CADET uses behavior-preserving transformations to generate new indices for retrieving previously inaccessible cases. These transformations are mathematically correct according to the *Chain Rule for Derivatives* for function composition. For example, an influence between two variables can be elaborated in series or in parallel, and each time an influence is elaborated a new variable is introduced (hypothesized) as shown in figure 4-27.

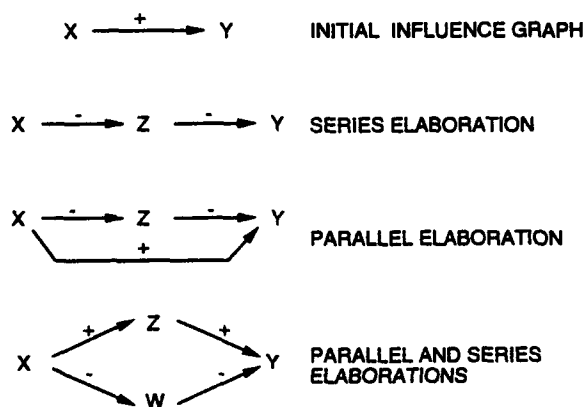


Figure 4-27: Series and parallel elaborations in behavior influence graphs

The influences implied by domain laws may be used to elaborate a given goal. For instance, assume it is our goal to achieve the influence: $X \rightarrow Z$. Also assume that there are no designs

that can achieve this effect directly. If, however, there is some domain principle which states that a quantity U influences Z , then the goal may be achieved by having X influence U . The goal is hence elaborated to: $X \longrightarrow U \longrightarrow Z$. This influence graph is used as a new index into the casebase. If cases or parts of cases with influences that match the new goal are found, they are retrieved and used.

4.17 Physical Synthesis of Design Cases

Sycara, Navin Chandra and Narasimhan [46, 65] have proposed the following steps for case-based design. These steps do not have to be followed sequentially, but are interleaved during problem solving.

1. *Development of a Behavioral Description.* The behavior of the desired artifact is described by qualitative relations (influences) describing how the inputs and outputs are related.
2. *Retrieval of Cases.* A collection of design cases or case parts whose behavioral descriptions bears similarities to the specified behavioral description is identified and retrieved. Retrieval is performed using not only the existing features of the input specification, but also indices arising from index transformations.
3. *Development of a Synthesis Strategy.* During this stage a description of how the cases and case pieces will fit together is developed.
4. *Physical Synthesis.* Realization of the synthesis strategy at the physical level. During physical synthesis the connecting elements between the retrieved components and sub-assemblies are determined. This entails taking into account possible interactions between components and sub-assemblies as well as reasoning about the dynamics of the system. Moreover, physical adaptations of the cases, needed to satisfy the desired behavior, are identified.
5. *Verification.* Undesirable interactions not identified during physical synthesis could lead to non-conformance of the design to the desired specifications. Qualitative and quantitative simulation are used during this phase.
6. *Debugging.* Explanations for non-conformance to the desired specifications are identified. This step involves a process of asking relevant questions and modifying the cases based on a causal explanation of the bug.

Previous work on case-based design for the CADET system has focused mainly on the first two steps. In this paper we investigate and propose a methodology for developing a synthesis strategy (step 3) and for physical synthesis of components and sub-assemblies retrieved from a casebase (step 4).

Types of Connections and Connectivity Graphs

Behavior influence graphs depict the dynamics of the systems they represent. The behavior preserving elaboration schemes that have been developed for the CADET system ensure correctness at the level of device behavior. The influences in a BIG between behavior variables (which are selected inputs, outputs and state variables) allow the qualitative simulation and

analysis of the dynamics of the related system. Labels on the edges (+, -, 0) provide first-order information about variable relationships. Higher-order information can also be included in this representation, although different elaboration rules have to be used to reason in such a context.

A drawback, however, of using an abstract representation is that a synthesis that looks acceptable at a high level of abstraction might fail to operate at a lower level. For example, at a high level of abstraction, a seesaw like device produces up and down motion. If this behavior is required in some context, the seesaw would be a candidate for synthesis. If we were to now look at the behavior of the seesaw in more detail, we would find that the ends of the seesaw move in an arc, not just up and down. To connect the seesaw to other components will require special connections that take into account the arc motion. The aim of this work is to provide and implement a methodology for the selection of appropriate connections from a casebase of connections.

A behavior influence graph also renders connectivity information between the components and sub-assemblies that have been retrieved after matching individual influences or groups of influences against the casebase of designs. Consider, for instance, the behavior influence graph elaborated from a specification for a hot-cold water faucet which allows independent control of the temperature and flow rate of water (figure 4-28). The variables in this graph are: displacement signal for temperature (S_T), displacement signal for flow rate (S_F), displacements (X_C and X_H), flow rates (Q_C , Q_H , and Q_M), and temperature of the mix (T_M). Figure 4-28 also shows the sets of influences that were matched to retrieve the various components: seesaw, tap-1, tap-2 and T-pipe. Interactions between these components are apparent from their sharing of behavior variables. Specifically, seesaw and tap-1 interact through X_C , seesaw and tap-2 through X_H , tap-1 and T-pipe through Q_C , and tap-2 and T-pipe through Q_H . Moreover, by observing the nature of the interacting variables, we can conclude that the connections between the taps and the seesaw have to be *kinematic*, since they impose constraints on kinematic variables¹¹, displacements. On the other hand, the connections between the taps and the T-pipe are *hydro-kinematic*, since they impose constraints on fluidic variables, the flow rates.

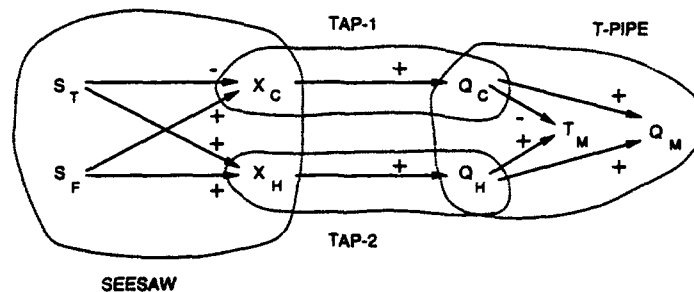


Figure 4-28: Behavior influence graph for hot-cold water faucet

Accordingly, we have identified the following four types of connections:

¹¹Kinematic variables are those representing translation or rotation.

1. *Kinematic Connections.* Their function is to transmit a force or moment. As a result they restrict the mobility of the connected parts, translationally or rotationally. Revolute, prism and spherical joints, and screw pairs are examples of this type of connection.
2. *Hydro-kinematic Connections.* Their function is to transport a fluid or transmit a hydraulic pressure. They may or may not restrict the mobility of the connected parts. Flexible and rigid pipes, and hollow kinematic connections having a passage for fluid are examples of this type of connection.
3. *Electro-kinematic Connections.* Their function is to transport an electric current or transmit a voltage. They may or may not restrict the mobility of the connected parts. A flexible electric wire and kinematic connections made of an electro-conductor material are examples of this type of connection.
4. *Thermo-kinematic Connections.* Their function is to transfer thermal energy. They may or may not restrict the mobility of the connected parts. Kinematic connections made of thermo-conductor material are examples of this type of connection.

Basically, hydro- electro- and thermo-kinematic connections are kinematic connections with the properties of being hollow to transport a fluid, or made of an electro- or thermo-conductor material, respectively. Hence, these three types of connections can be easily realized by adapting kinematic connections. In the remainder of this paper we will deal with the selection of kinematic connections which can be easily modified to provide, if necessary, one or more of the other three functions specified above.

Since we are interested in the synthesis of devices which transmit or control relative movement, we are mostly concerned with rigid bodies connected together by joints. Usually the joints are formed by simple contact between adjacent bodies, though occasionally devices contain joints which are flexible, whether by belt or band, by springs or some other elastic component. The parts of two components of a device which are connected together form a kinematic pair. Kinematic pairs are classified according to the following characteristics [53]:

- **Type of motion.** The motion of a point on one element relative to the other may be a space, surface or line motion.
- **Number of degrees of freedom.** A joint may restrict 1, 2, 3, 4 or 5 of the degrees of freedom of the pair of elements¹².
- **Type of contact.** This may be surface, line or point contact.
- **Type of closure.** In a self-closed kinematic pair contact between elements is maintained by construction. On the other hand, in the case of a force-closed kinematic pair a force is necessary to maintain contact.
- **Lower pairs:** There is surface contact between elements, so they are used to transmit high loads. These are some of the lower pairs (see figure 4-29):
 1. *Spherical pair* allows three degrees of freedom.

¹²A free rigid body has six degrees of freedom.

2. *Planar pair* allows three degrees of freedom.
 3. *Cylindrical pair* allows two degrees of freedom.
 4. *Revolute-prism pair* allows two degrees of freedom.
 5. *Slotted sphere pair* allows two degrees of freedom.
 6. *Turning or revolute pair* allows one degree of freedom.
 7. *Prism or sliding pair* allows one degree of freedom.
 8. *Screw or helical pair* allows one degree of freedom.
- **Higher pairs.** There is line or point contact between elements, so they provide low-friction connections. They constitute the basis for gears, roller-bearings and cams. These are two higher pairs (see figure 4-30):
1. *Cylinder-sphere pair* allows four degrees of freedom.
 2. *Plane-sphere pair* allows five degrees of freedom.
 3. *Plane-cylinder pair* allows four degrees of freedom.

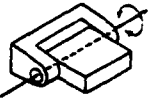
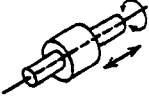
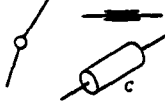
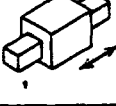
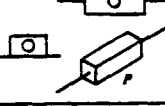
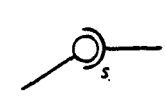
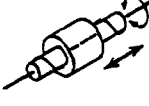
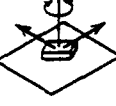
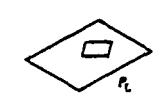
Name of Pair	Geometric Form	Schematic Representations	Degree of Freedom
1. Revolute (R)			1
2. Cylinder (C)			2
3. Prism (P)			1
4. Sphere (S)			3
5. Helix (H)			1
6. Plane (P _L)			3

Figure 4-29: Lower kinematic pairs

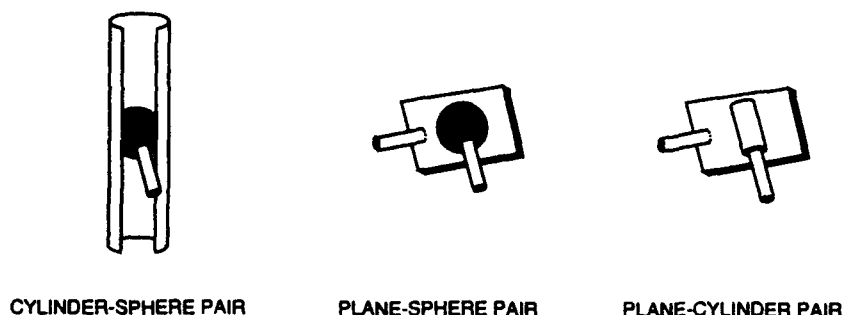


Figure 4-30: Higher kinematic pairs

Connectivity information given by a behavior influence graph can also be represented by means of a *connectivity graph*. The nodes in a connectivity graph represents the various retrieved components or sub-assemblies, while the edges indicate the existence of some type of connection. For example, figure 4-31 shows the connectivity graph for the hot-cold water faucet of figure 4-28. As mentioned above, the connections between the seesaw and the taps are kinematic, thus they restrict one or more degrees of freedom. However, the hydro-kinematic connections between the taps and the T-pipe may or may not restrict degrees of freedom, that is, a flexible pipe could be used to connect these components.

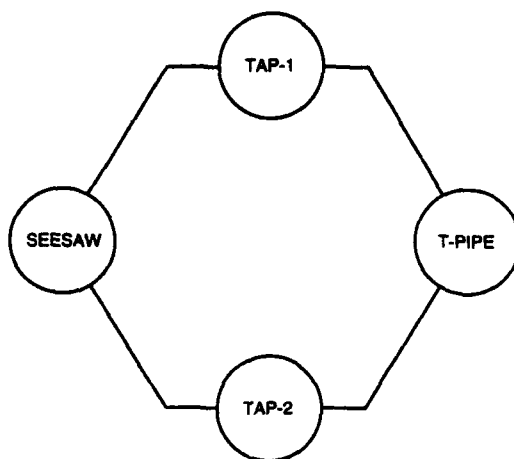


Figure 4-31: Connectivity graph for hot-cold water faucet

4.18 Selection of Connections

As mentioned above, appropriate connections between components and sub-assemblies are essential to ensure the correct behavior of a synthesized device. Kinematic, hydro-kinematic, electro-kinematic and thermo-kinematic connections make possible for the different parts of an artifact to interact in these four domains so their assembly achieves the required behavior. Conversely, a wrong type of connection generates undesirable interactions among components. In what follows we will focus on those interactions that determine the mobility of the parts. That is, we will present connectivity conditions that a device have to fulfill to ensure that it is kinematically viable and correct.

The augmented connectivity graph shown in figure 4-32 represents the components, and their connectivities, retrieved after matching a behavior influence graph. "dof-*i*" for the component or sub-assembly "comp-*i*" indicates its degrees of freedom (mobility) when taken in isolation, i.e., disconnected from the rest of components. A rigid body has six degrees of freedom in general motion and three degrees of freedom in plane motion. A sub-assembly, constituted of two or more connected parts, can have more than six degrees of freedom in general and more than three in plane motion. For example, in general, a tap has seven degrees of freedom since six coordinates are necessary to specify the position of the tap in space and one additional parameter is needed to characterize the variable aperture of the tap. In figure 4-32, "rdof-*ij*" for the connection between components "comp-*i*" and "comp-*j*" indicates the number of degrees of freedom restricted by such connection. For example, a spherical connection restricts three degrees of freedom.

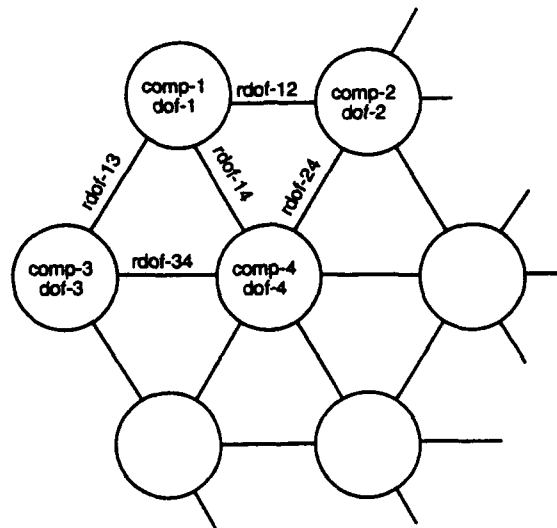


Figure 4-32: A generic connectivity graph

4.18.1 Kinematic Inputs

An artifact is a system, that is, a collection or interconnection of functional units which interact with each other and with an environment to perform some purposeful behavior. As a system, it is possible to distinguish among inputs (excitations), outputs (responses) and state variables. Inputs (outputs) are associated with those nodes of a behavior influence graph which have in-degree (out-degree)¹³ zero. Such is the case of the input signals S_T and S_F in the behavior influence graph for the hot-cold water faucet of figure 4-28. Some of the inputs to a system could be translational or rotational signals, while others could be electric, hydraulic or thermal signals.

Kinematic inputs are the rotational or translational signals required to define the configuration of

¹³The in-degree (out-degree) of a node is equal to the number of edges incident into (out of) the node.

a device when the non-kinematic components (fluids, electric current and heat flux) have been removed from the connecting elements. That is, when only the kinematic function of the hydro-electro- and thermo-kinematic connections is taken into account. Consider, for example, the sheet-metal bending machine of figure 4-33 whose behavior influence graph is shown in figure 4-34. We note from the behavior influence graph that this device has one input, S_X , and one output, Y . However, from a kinematic point of view this device requires two parameters (kinematic inputs) to define its configuration, one for the taps, seesaws, rigid body and frame sub-assembly, and the second for the piston, cylinder and frame sub-assembly. These two sub-assemblies interact by means of a medium that is not a rigid body but a fluid.

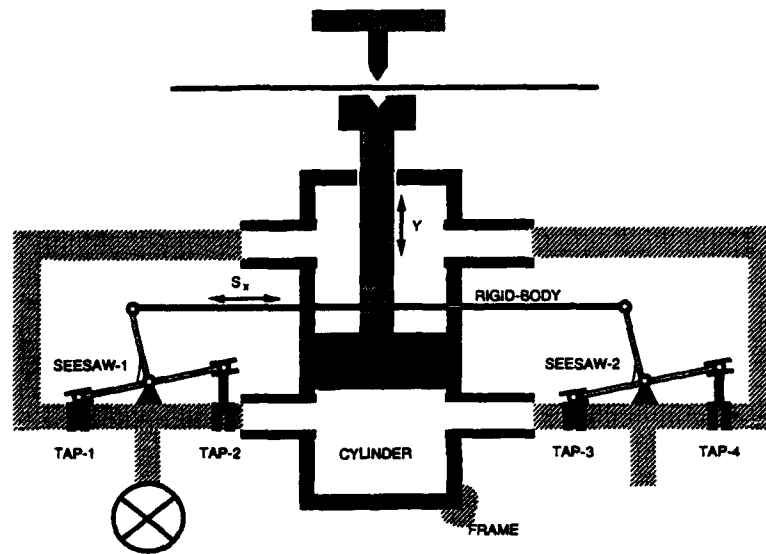


Figure 4-33: Sheet-metal bending machine

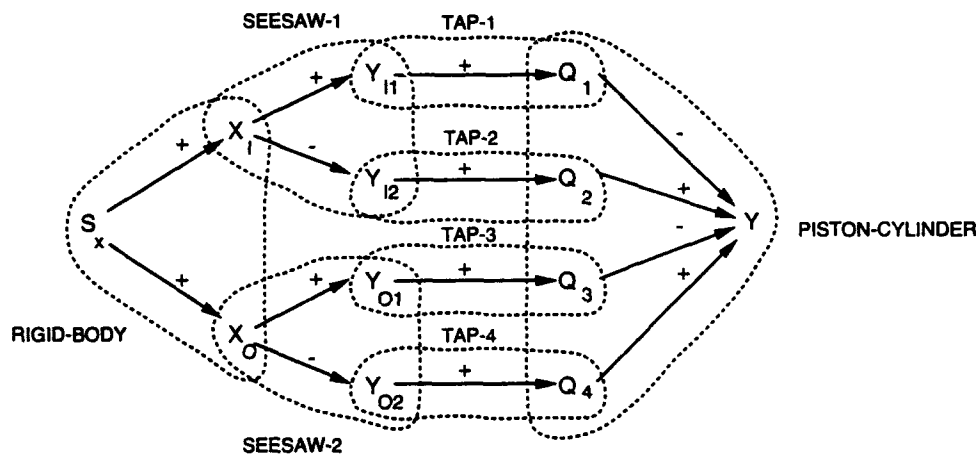


Figure 4-34: Behavior influence graph for sheet-metal bending machine

Let G be the behavior influence graph of a device and let G^* be the resulting behavior influence graph after removing non-kinematic variables from G . Then the number of kinematic inputs is equal to the number of kinematic variables (nodes) in G^* with in-degree zero. Also, the number

of kinematic inputs is equal to the number of degrees of freedom of the device. Figure 4-35 shows the influence graph of kinematic variables for the sheet-metal bending machine of figure 4-33. As expected, it contains two nodes with in-degree zero; therefore, this device has two degrees of freedom.

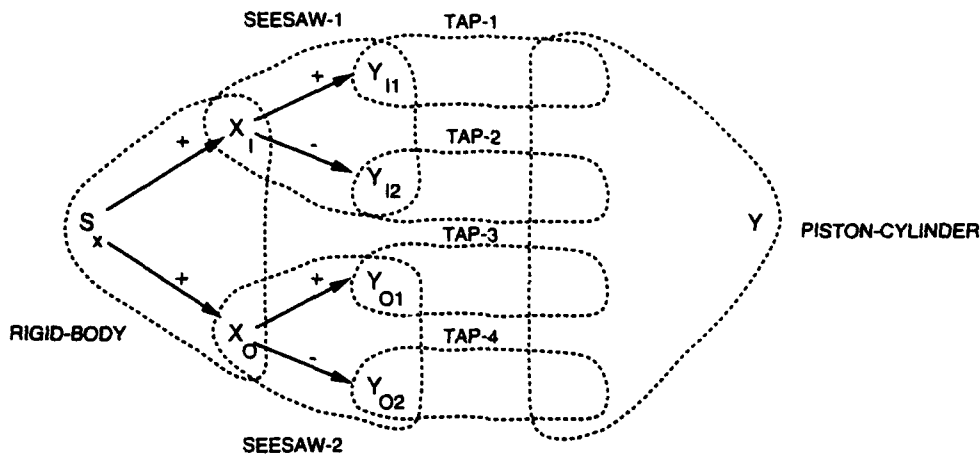


Figure 4-35: Behavior influence graph for sheet-metal bending machine containing only kinematic variables

If G_i^* is a connected component of G^* , then the number of degrees of freedom of the sub-assembly (subgraph) associated with the connected component in the behavior influence graph¹⁴ is equal to the number of kinematic variables (nodes) in G_i^* with in-degree zero. The metal bending machine has two kinematically independent sub-assemblies: taps, seesaws, rigid body and frame; and piston, cylinder and frame sub-assembly. Each of them has one degree of freedom.

4.18.2 Connectivity Conditions

Selection of the appropriate connections between components and sub-assemblies is based on the number of degrees of freedom (dof's) that the connections restrict. Thus, a spherical or planar pair is chosen if three dof's have to be restricted; a cylindrical, revolute-prism or slotted sphere pair is chosen if four dof's have to be restricted; a cylinder-sphere pair is chosen if two dof's have to be restricted; and so on. These connections are retrieved from a casebase that contains those mentioned in section 4.17.

Given a connectivity graph of the retrieved components and sub-assemblies composing a device, the first step for the selection of connections is to augment the connectivity graph by adding a node with zero degree of mobility that represents the *frame* of the device. This step is necessary since static elements are not retrieved by a case-based design system such as CADET which

¹⁴A sub-assembly associated with a connected component in the behavior influence graph will be called a *kinematically independent sub-assembly*.

represents behavior based on the dynamic characteristics of the design cases. The next step is to assign to each edge of the connectivity graph the number of degrees of freedom, $rdof_{ij}$, that the associated connection would restrict. The assigned restricted degrees of freedom for each edge of the connectivity graph have to satisfy the following conditions. These conditions are based on the basic mobility criterion of Grübler [15] and Malysheff [37] (see appendix).

1. The number of degrees of freedom of the device is equal to

$$\sum_{\text{nodes}} dof_i - \sum_{\text{arcs}} rdof_{ij}$$

2. The number of degrees of freedom of a kinematically independent sub-assembly (subgraph) associated with a connected component in a behavior influence graph is equal to

$$\sum_{\substack{\text{nodes} \\ \text{subgraph}}} dof_i - \sum_{\substack{\text{arcs} \\ \text{subgraph}}} rdof_{ij}$$

3. Edges that depict kinematic connections restrict at least one degree of freedom since, by definition, kinematic connections restrict the mobility of the connected parts.
4. A sub-assembly (subgraph) cannot have negative degrees of freedom. Otherwise the design is either wrongly over-constrained, or possesses general or overclosing constraints which have to be considered during detailed design (see appendix). The number of degrees of freedom of a sub-assembly is equal to

$$\sum_{\substack{\text{nodes} \\ \text{subgraph}}} dof_i - \sum_{\substack{\text{arcs} \\ \text{subgraph}}} rdof_{ij}$$

5. A sub-assembly (subgraph), with only non-zero $rdof$'s and containing an *input component*¹⁵ and *frame* has a number of degrees of freedom greater than or equal to the number of kinematic *input variables* of the input component. This condition ensures the required degree of mobility of the input component(s).
6. A sub-assembly (subgraph), with only non-zero $rdof$'s and containing an *output component*¹⁵ and *frame* has a number of degrees of freedom greater than or equal to the number of kinematic *output variables* of the output component. This condition ensures the required degree of mobility of the output component(s).
7. A node a (an assembly), with M neighboring nodes b_i , can substitute for N nodes a_i (its sub-assemblies) as long as

$$dof_a = \sum_{i=1}^N dof_{a_i} - \sum_{\substack{i,j=1 \\ i \neq j \\ i < j}}^N rdof_{a_i a_j}$$

and

¹⁵An input component is associated with a subgraph (of a behavior influence graph after removing non-kinematic variables) that has kinematic variables (nodes) with in-degree zero. An output component is associated with a subgraph that has kinematic variables (nodes) with out-degree zero.

$$\text{rdof-}ab_j = \sum_{i=1}^N \text{rdof-}a_i b_j$$

as shown in figure 4-36. Conversely, N nodes making up an assembly can substitute for the single node representing the assembly.

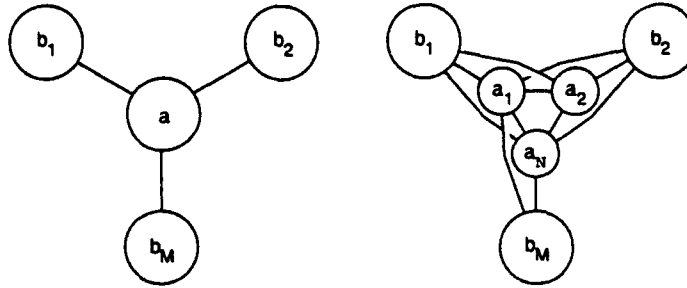


Figure 4-36: Connectivity graphs for an assembly and its components

8. After removing the frame, the number of degrees of freedom of a device increases by at least *six* (or *three* for planar devices). This condition depicts one of the functions of the frame, which is to provide a static reference system for the specification of the behavior of the device.

The above equations, defining the number of degrees of freedom of a device as the difference of the freedoms of the individual components and the freedoms restricted by their connection, are based on the mobility equation for a mechanism of Grübler [15] and Malysheff [37]. This equation, however, is not a sufficient criterion in general. Grübler-Malysheff's equation is not correct under certain geometric configurations that generate redundant and passive freedoms, and overclosing and general constraints. These deviations from the basic mobility criterion are the result of the geometric arrangement of the parts in the final assembly of a device. Specifically, they are due to parallelism and/or collinearity of axes of motion, constraints on dimensional parameters, and less frequently topology of the artifact. Their presence, in general, entails strict tolerances and manufacturing specifications, and the slightest deviation from such specifications results in a non-operating device. Case-based design, however, enables the designer to choose final configurations that do not exhibit such deviant behaviors. We describe in the appendix these particular configurations.

4.19 Example

Consider the design of the hot-cold water faucet whose behavior influence graph and connectivity graph were shown in figures 4-28 and 4-31, respectively. Following the criterion given in section 4.18.1, this device has two degrees of freedom. Figure 4-37 shows the same connectivity graph augmented with the frame node and the degrees of freedom of each component (frame, 0-dof, seesaw, 3-dof, and T-pipe, 3-dof) and sub-assembly (taps, 4-dof). The connection between the body and the gate of the taps is known to be a sliding pair, which restricts two degrees of freedom in plane motion (five degrees in general motion). The seesaw is the input component with two kinematic input variables, S_T and S_F , while the taps are output components with one kinematic output variable each, X_C and X_H .

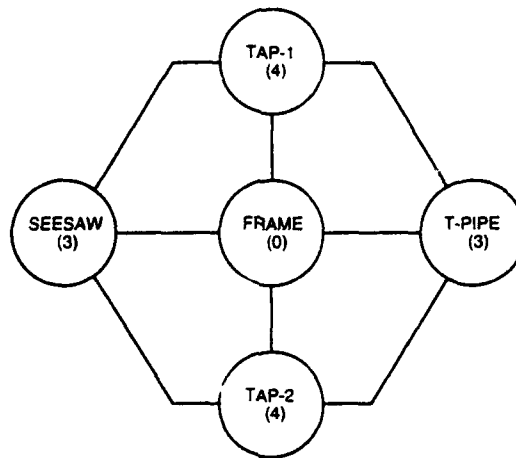


Figure 4-37: Augmented connectivity graph for hot-cold water faucet

Although the principles established in section 4.18.2 as connectivity conditions apply to general motion, we have restricted this example to plane motion. Figure 4-38 shows the lower pairs for plane motion, including also the rigid connection which restrict three degrees of freedom. Figure 4-39 shows the components of the faucet retrieved by the case-based reasoner and a collection of connecting elements from where the connections will be selected.

TYPE	RESTRICTED DEGREES OF FREEDOM	RESTRICTED TRANSLATIONS	RESTRICTED ROTATIONS
RIGID	3	2	1
REVOLUTE	2	2	0
SLIDER	2	1	1
ROLL-SLIDER	1	1	0

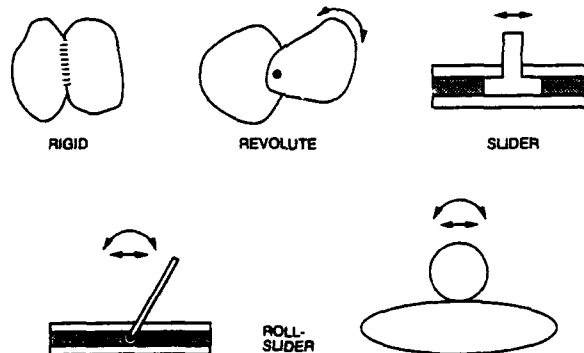


Figure 4-38: Lower pairs for plane motion

A program that generates sets of connections which verify the eight connectivity conditions of section 4.18.2 has been implemented. The connections for the hot-cold water faucet were chosen

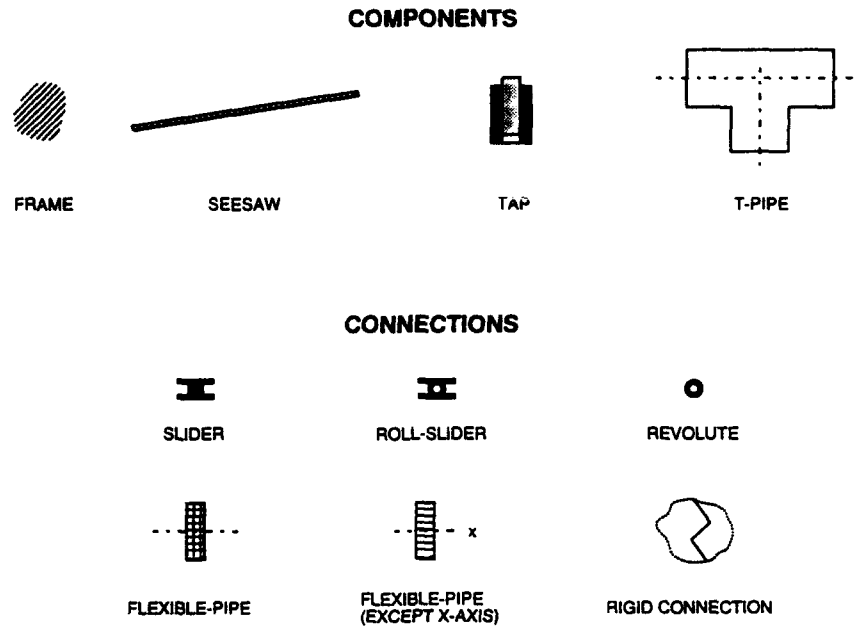


Figure 4-39: Components and connections used in the design of hot-cold water faucet

from those shown in figure 4-39 which also satisfy the following conditions:

- SEESAW & TAP-1: Connections that restrict one or two degrees of freedom.
- SEESAW & TAP-2: Connections that restrict one or two degrees of freedom.
- T-PIPE & TAP-1: Connections that restrict none, one or three degrees of freedom.
- T-PIPE & TAP-2: Connections that restrict none, one or three degrees of freedom.
- SEESAW & FRAME: Connections that restrict none or one degree of freedom.
- T-PIPE & FRAME: Connections that restrict two or three degrees of freedom.
- TAP-1 & FRAME: Connections that restrict node or two degrees of freedom.
- TAP-2 & FRAME: Connections that restrict none or two degrees of freedom.

Thirty-six different feasible configurations were generated from a total of 576 ($2^4 \times 3^2$) possible configurations for the given space of connections. Figure 4-40 shows the connectivity graphs and physical configurations for five of these feasible configurations. These configurations have the same topology, but they differ from each other either in the election of the connecting elements or in whether or not the components are connected to the frame.

4.20 Conclusions

In this paper we have presented a methodology for physical synthesis of design components and sub-assemblies retrieved under a case-based design framework. Indexing mechanisms for the connecting elements between the retrieved parts were developed based on mobility conditions defined by design specifications. The goal of this work was to provide to a case-based design

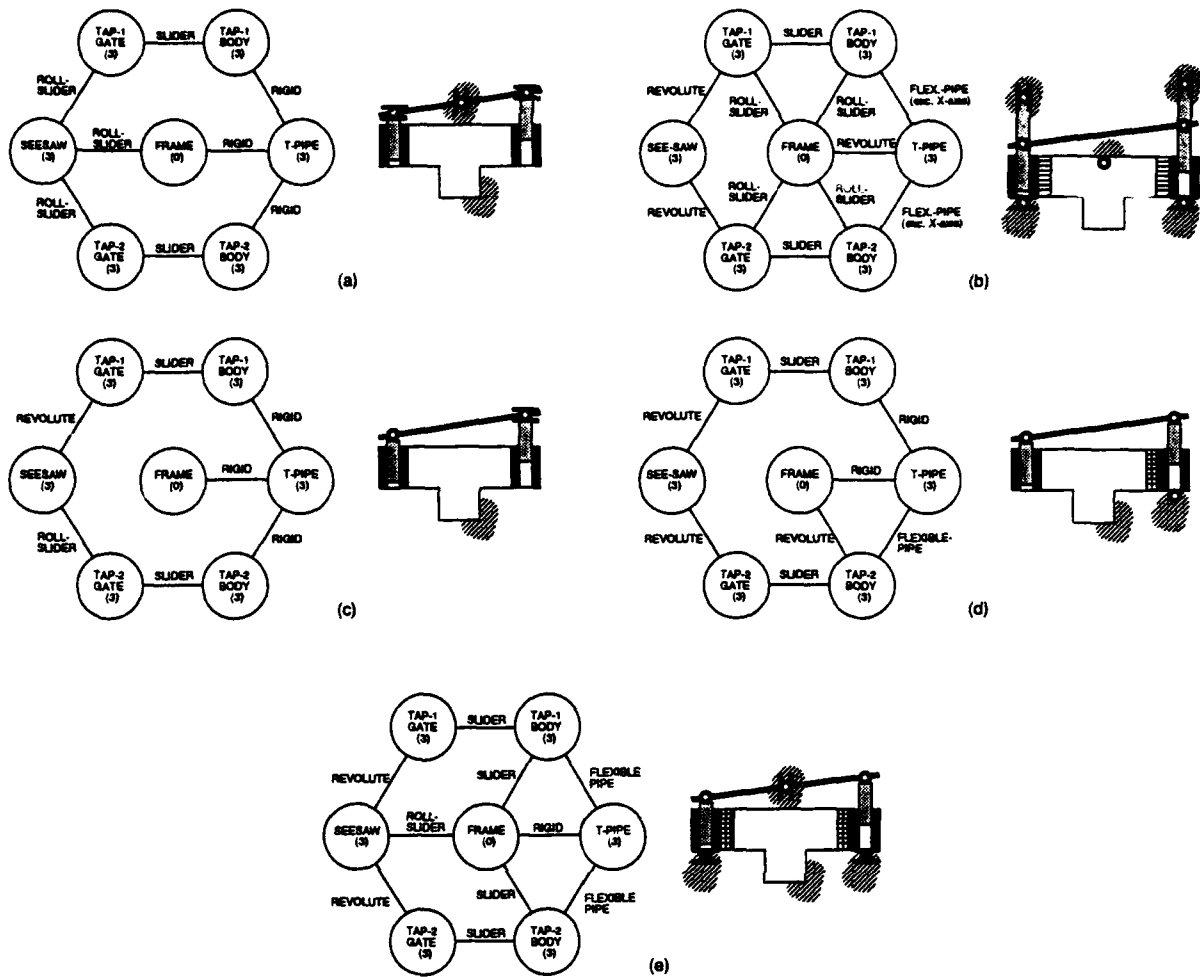


Figure 4-40: Five feasible configurations for hot-cold water faucet

system with reasoning mechanisms for design synthesis at the configuration level. Multiple feasible assembly modes for the design parts enlarge the universe of design alternatives to be presented to the design engineer. The information necessary to accomplish this task is still of a high-level nature, namely, the topology of the artifact and its abstract behavior specification. However, constraints on the final geometric arrangement of the parts could be a side-product of this stage. Verification of the final configuration to detect and reason about dimensional and geometric constraints is a pending issue for further study and implementation in our system.

Appendix: Dimensional and Geometric Constraints in Physical Synthesis

As mentioned in section 4.18.2, the mobility criterion on which the given connectivity conditions were based is not valid for general configurations. The most general mobility equation is given by Bagci in [3, 4] as

$$F_i = F_0 + M + F_c - F_r - F_p$$

where F_i is the number of (kinematic) inputs required to drive the mechanism (equal to the number of degrees of freedom).

F_0 is the basic mobility given by Grübler [15] and Malysheff [37] as

$$F_0 = q(n-1) - \sum_{\substack{\text{dof's of} \\ \text{pairs}}} (q-i) N_i$$

where q is the number of the components of the general rigid body displacement (three rotations and three translations) that can take place in the space of motion of the mechanism. $q = 6$ in general rigid body motion and $q = 3$ in plane motion. n is the number of links (rigid components) including the frame. N_i is the number of pairs permitting i degrees of motion for one link relative to the adjacent one.

M is the total number of *general constraints*. It is equal to the sum of the number of general constraints (the number of non-existing components of the general rigid body displacement) in each loop. Figure 4-41 shows a planar mechanism with *one* general constraint on rotation about the axis normal to the plane of motion.

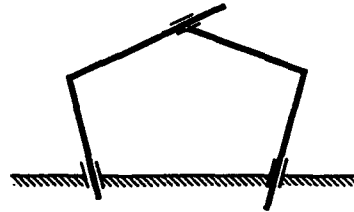


Figure 4-41: Mechanism with one general (rotational) constraint

F_c is the total number of *overclosing constraints* due to geometric and dimensional constraints in a single loop¹⁶, in a group of loops, and in each gear train loop.

$$F_c = F_{cs} + F_{cl} + F_{cg}$$

where

$$F_{cl} = \sum_{\substack{\text{group of} \\ \text{loops}}} (K_{cj} - 1)$$

and K_{cj} is the number of loops in the j^{th} group that exhibits overclosing constraints. Figure 4-42 shows a planar mechanism with *two* overclosing constraints due to parallelism between links.

F_p is the total number of *passive freedoms*. These are freedoms that due to geometry and topology¹⁷ the mechanism never experiences. For example, the spherical pair of the CSC mechanism shown in figure 4-43 has its rotational freedom about the axis normal to the α, β -plane passive.

F_r is the total number of *redundant freedoms*. These are freedoms of motion of some

¹⁶Bennett's 4R four-bar mechanism, spherical 4R mechanisms and Bricard's 6R six-bar mechanism are examples of mechanisms showing single loop overclosing constraints.

¹⁷The topology of a device is defined by its components and their connections.

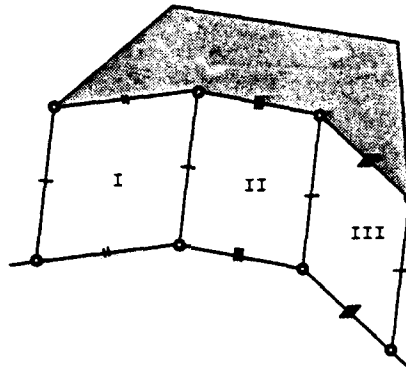


Figure 4-42: Mechanism with two overclosing constraints

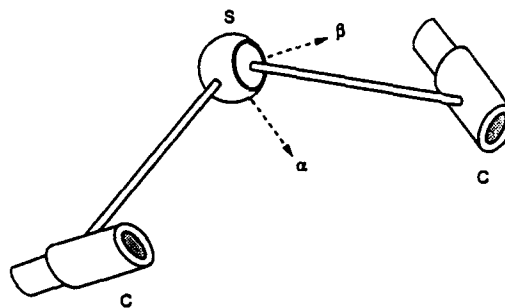


Figure 4-43: Mechanism with one passive freedom

components which do not affect the input and output motion of the whole assembly. For example, the RSSR mechanism shown in figure 4-44 experiences a redundant freedom of motion about the axis of the coupler link connecting the two spherical pairs. Redundant rotational freedoms occur when the axes of rotation of pairs (spherical, cylindrical or revolute pairs) belonging to the same link are collinear. Redundant translational freedoms occur when the axes of translation of pairs (prism, cylindrical, planar or revolute-prism pairs) belonging to the same link are parallel.

Overclosing and general constraints are the result of the geometric arrangement of the parts in the final assembly of a device. Specifically, these constraints are due to parallelism or collinearity of axes of motion and constraints on dimensional parameters. Their presence, in general, entails strict tolerances and manufacturing specifications, and the slightest deviation from such specifications results in a non-operating device. The mechanism shown in figure 4-42, for example, would get stuck if "perfect parallelism" were not achieved between members. A robust design should not depend on stiff tolerances and precision engineering to fulfill its required function. Moreover, wear on parts and joints affects dimensional parameters and introduces misalignment errors and undesirable forces and behaviors. Thus, a device configuration generated by our system, which lacks general and overclosing constraints ($M = F_c = 0$), is expected to be more robust than another that relies on those constraints to achieve a behavioral requirement. Verification of the final configuration to detect and take into

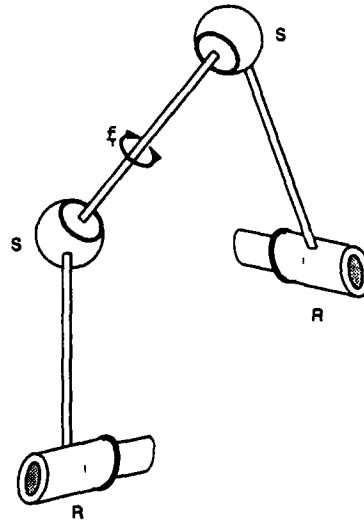


Figure 4-44: Mechanism with one redundant freedom

account overclosing and general constraints is needed and its automation constitute a pending issue for implementation in our system.

Passive freedoms result from not only the geometric arrangement of the parts but also the topology of the artifact, as depicted by the CSC mechanism of figure 4-43. In this example, there is no dimensional constraint or geometric configuration that eliminates the passive freedom of the spherical joint. As with overclosing and general constraints, detection of and reasoning about passive freedom have not been fully implemented in our system.

Redundant freedoms occur under very specific circumstances which have been defined above. Redundant freedoms are not present as long as collinearity or parallelism of axes of rotation and translation of pairs on the same member is avoided. For example, in the design shown in figure 4-40(e), the sliding connections between the tap's bodies and frame, and the tap's bodies and gates were not placed parallel to avoid introducing a redundant freedom. In practice, however, redundant freedoms, and also passive freedoms, may be kept in the mechanism for operation and lubrication. This entails their appropriate identification and consideration in the mobility equation.

The previous considerations can be used to adapt a device to perform certain function. For example, a rigid five-member structure similar to that shown in figure 4-42, whose function would be merely to support a vertical load, could be adapted, by imposing geometric and dimensional constraints, for a rigid-body guidance application. Moreover, reasoning about geometric and dimensional features of a device, in addition to about its structure, enables a case-based design system to generate more configurations for a given specification.

Chapter 5

Representation and Reasoning about Performance

5.21 Introduction

Case-Base Reasoning (CBR) [30] is the problem solving paradigm where previous experiences are used to guide problem solving. Cases similar to the current problem are retrieved from memory, the best case is selected from those retrieved and compared to the current problem. The precedent case is adapted to fit the current situation, based on the identified differences between the precedent and the current case. Successful cases are stored so they can be retrieved and re-used in the future. In this way, learning is integrated with problem solving. Failed cases are also stored so that they will warn the problem solver of potential difficulties and help recover from failures. If a current case recalls a past failure, then the problem solver is warned not to attempt the failed solution.

The connection between case-based reasoning and traditional engineering design is clear and well studied [46, 47, 67, 65, 71]. Engineering designers often combine parts of different design cases to devise an artifact that satisfies a useful need [59, 75, 76]. For design, case retrieval is done based not only on geometric features but also on abstract descriptions of the device function and behavior¹⁸, as well as on its performance and topology. The parts of an artifact are often highly-integrated and tightly-coupled to render the required function. Although there is no simple and clear correspondence between the functional characteristics of a device and those of its components, the overall behavior of the device can be decomposed into sub-behaviors of its components¹⁹, as shown by Sycara and Navin Chandra in their work on case-based design [46]. They use a graph-based representation of behavior and apply behavior-preserving transformations to an abstract description of the desired behavior until a description is found that

¹⁸A device's *behavior* is what it does, while its *function* is what it is used for. For example, a clock has the behavior of moving its hands, while it has the function of telling time. The same function can be achieved through different behaviors. For example, digital and analog clocks tell time.

¹⁹This is achieved by transforming a given behavior specification into alternative decomposable forms that preserve the overall desired behavior

closely corresponds to some collection of relevant cases. This approach allows for retrieval of cases without imposing a predetermined decomposition of the design and it also utilizes knowledge of domain laws. Innovative designs result when the design parts or components are taken from design cases that are functionally dissimilar to the current design problem.

Frequently, high-level qualitative descriptions of function or behavior are not appropriate for retrieval and synthesis of designs which have to perform according to a low-level numerical specification. This situation demands for a behavior representation that allows for reasoning at the numerical level while still eases case indexing and retrieving when less important details are left ignored. The aim of this paper is to present a representation of behavior that includes performance information such as ranges of operation and types of response to various input signals. The high-level behavior representation and reasoning mechanisms used in a case-based design tool such as CADET [46] is used as foundation for developing the lower level abstractions and representations needed for reasoning about the performance of a device and performance-based retrieval of design cases.

This paper is organized as follows. The next section provides an overview of CADET, a case-based design tool: specifically, the first-order behavior influence representation. We then present an extension of the behavior representation in CADET which takes into account second-order information. This is followed by a presentation on behavior representation and reasoning which includes low level performance information...

5.22 Behavior Representation in CADET

Sycara, Navin Chandra and Narasimhan have developed the CADET (CAsE-based DEsign Tool) system, one of the first tools for automated synthesis of mechanical designs that successfully exploits prior designs for reuse in other contexts. CADET takes as input a high level behavior specification of a desired artifact and retrieves prior designs or design parts whose composition delivers the required behavior. CADET's behavior-based indexing algorithms, composition schemes, and reuse methods have been extensively reported and referenced in the literature [46, 47, 64, 67, 66, 65, 69, 70, 71]. CADET has access to a case memory and engineering domain laws and principles. Each case is represented in terms of a multi-layered representation expressing function, behavior and structure of the device and relations between them. Abstractions that can be used for indexing a case are: (1) linguistic descriptions, (2) functional block diagrams, (3) behavior influence diagrams, (4) qualitative states, (5) structural features, and (6) performance features. This paper will focus primarily on the integration of performance features with the influence diagram representation of behavior.

In CADET, the behavior of a device is represented as a collection of influences between the various behavior variables²⁰. In the following, we briefly review and illustrate the first-order

²⁰Behavior variables are those inputs, outputs and state variables chosen to characterize a system's behavior at a high level of abstraction.

behavior representation used in CADET and the process of index elaboration and matching. In the next section, second-order influences are presented as an extension to the CADET representation. Second-order information allows for reasoning at lower levels of abstraction, hence closer to levels of abstraction used to describe performance.

First-Order Behavior Influence Representation

A first-order influence is a qualitative differential relation between two variables, one of which is considered the independent variable and the other the dependent variable. Behavior influences are based on the notion of confluences [6] and causality [24]. Causality assignments determine the dependency and independence of the behavior variables hence the direction of the corresponding influences. In general, a behavior influence graph²¹ is a directed graph whose nodes represent behavior variables and whose edges have assigned qualitative values depicting the influence of one variables on another. The sign on an influence depicts the direction of variation of the dependent variable with respect to the independent one. For the case where there exists a mathematical relation between the independent and dependent variables x and y , i.e., $y = f(x, z)$, with f a differentiable function, the sign on the influence is given by the sign of the partial derivative of f with respect to x . In the following, we illustrate the process of elaboration of new behavior variables. For more details of the approach see [46].

Given an influence graph representing a required behavior, behavior-preserving transformations can be used to generate new indices for retrieving of previously inaccessible designs. These transformations are mathematically correct according to the *Chain Rule for Derivatives* for function composition. An influence between two variables can be elaborated in series or in parallel, and each time an influence is elaborated a new variable is introduced (hypothesized) as shown in figure 4-27. Influences implied by domain laws may also be used to elaborate a given goal. For instance, assume it is our goal to achieve the influence: $X \longrightarrow Z$. Also assume that there are no designs that can achieve this effect directly. If, however, there is some domain principle which states that a quantity U influences Z , then the goal may be achieved by having X influence U . The goal is hence elaborated to: $X \longrightarrow U \longrightarrow Z$. This influence graph is used as a new index into the casebase. If cases or parts of cases with influences that match the new goal are found, they are retrieved and used.

Consider the design of a device that controls the flow of water into a flush tank. The behavior of the flush tank can be specified as follows: *As the water level (D) in the tank increases, the rate of water flow into the tank (Q) should decrease.* This behavior specification is expressed in terms of the influence $D \longrightarrow Q$, and this influence is used as an index for retrieving cases. If no case can be found that matches a particular design specification, it will be desirable to find cases whose composed behavior is equivalent to the specified behavior.

²¹Unless otherwise stated, a behavior influence graph contains only first-order qualitative information

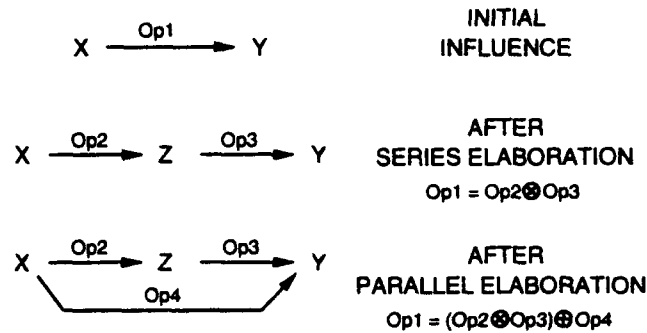


Figure 5-45: Series and parallel elaborations in first-order behavior influence graphs

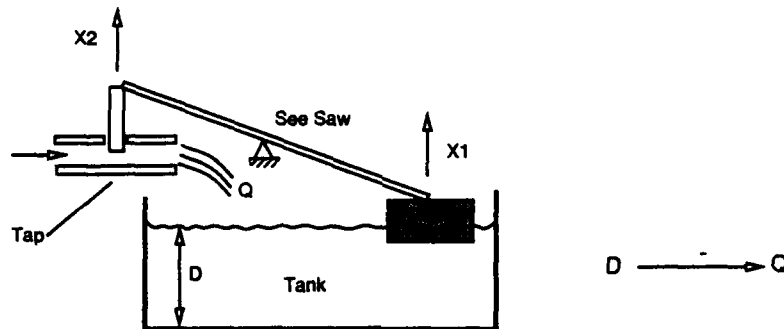


Figure 5-46: A flush tank and its behavior specification

Figure 2-8 shows a possible solution to the problem. Three cases have been identified (a float, a seesaw, and a tap) and correspondingly represented by the following influences: $D \xrightarrow{+} X1$, $X1 \xrightarrow{-} X2$ and $X2 \xrightarrow{+} D$. The combined effect of these influences is equivalent to that in the original specification. To find cases whose composition will deliver the specified behavior, the system first automatically decomposes the specified behavior into a behaviorally equivalent set of influences and then uses each influence as an index for matching precedents in the casebase.

The first-order influence behavior representation has the highest level of abstraction. The class of strictly monotonic relations between two variables is divided into two families, each associated with the positive or negative influence, as depicted in figure 5-47²². A high level of abstraction facilitates indexing and retrieving of design cases which behave according to a sub-graph of the behavior influence graph representing the required behavior. Retrieving is carried out by matching behavior variables and influence signs of the sub-graph. A drawback of a high level of abstraction is the lost of information on certain performance characteristics which are better defined by a more detailed representation of the relations between behavior and design variables. At the lowest level of abstraction these relations can be represented by algebraic and differential equations. However, equations cannot be used as indices, much less for efficient

²²Zero-first-order influence is not shown since it is associated with static behavior and only dynamic behavior will be treated in this paper.

retrieving of design cases. As a result, influences with intermediate levels of abstraction need to be explored as a means for representing and retrieving design cases which differ from each other not only by the monotonicities of their influences but also by other forms of lower level characteristics of the variables relations. Following we present these other forms for variable-dependency representation.

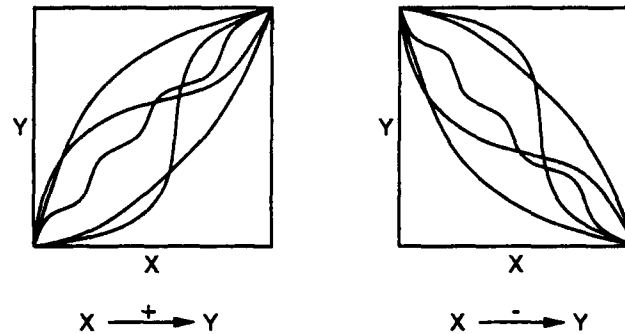


Figure 5-47: Positive and negative first-order influences

5.23 Second-Order Behavior Influence Representation

Higher-order information on the variation of the dependent variable with respect to an independent variable can also be represented in a behavior influence graph. For the case where there exists a mathematical relation between the related variables, the higher-order influences are given by the signs of the second, third, and so on, partial derivatives in addition to the sign of the first derivative (first-order influence). The highest the order of the representation, the lowest the level of abstraction. The number of indices for retrieving of design cases increases with the order of the representation.

In particular, when there is an implicit mathematical dependency, second-order influences describe the interaction of two variables in their influence on a third, determined by the sign of the mixed partial second-order derivative, or second-order properties of the relation between two variables, given by the sign of the univariate second-order partial derivative. The first type of second-order relationship is analogous to the concept of qualitative synergy found in qualitative probabilistic network [77] and it requires hyper-edges for its representation. The second type of relation has been proposed as an extension for monotonic influence diagrams [38, 39] and can be represented by arcs, as the first-order influences. Specifically, this level of abstraction allows for the description and representation of both monotonic and convexity properties of the dependency relations between behavioral parameters. Subsequent retrieval of design cases with the appropriately shaped response is then possible. Figure 5-48 shows the classes of monotonic dependencies that can be characterized with a second-order representation. Six families of variable dependencies can be identified according to the type of first- and second-order influence, the latter depicted by a boxed sign. They are a subset of the relationships represented solely on the basis of first-order influences, and have as additional constraint the invariance of the sign of their second-order derivatives. As a result *S-shaped* responses cannot be represented

nor retrieved.

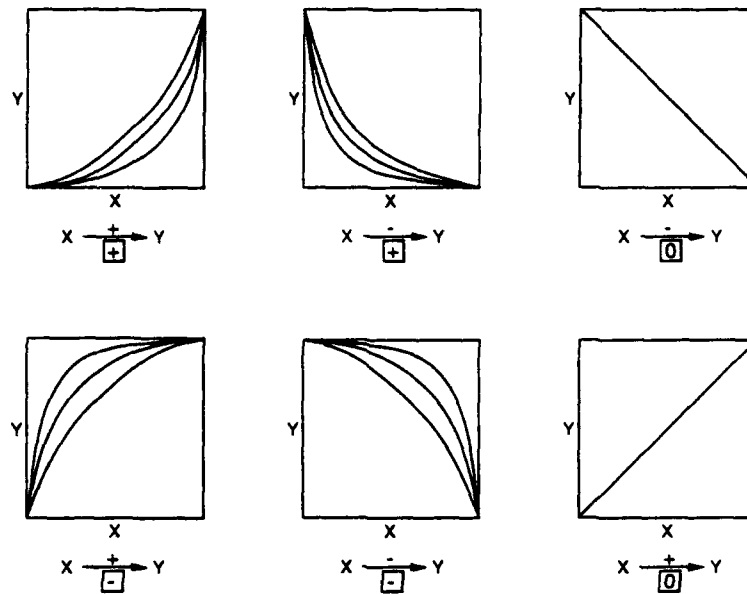


Figure 5-48: Second-order influences

As with first-order influences, higher-order influences can be transformed by behavior-preserving elaborations so new variables are introduced and new indices generated. Figure 5-49 shows series and parallel elaborations when both first- and second-order influences are used for behavior representation. Hyper-edges make sub-graph matching more difficult; however, they can be ignored by setting the corresponding second-order influence to zero (in the figure, $Op5' = 0$)

5.24 Representation and Reasoning about Performance

It is not enough to capture the qualitative behavior of a device. Information as to its performance (how well the device performs the specified behavior) is crucial. Different performance requirements for the same behavior specification may entail different designs. For example, although the basic behavior of all valves is the same, there are hundreds of valve types primarily differentiated by performance. Similarly, the type of a hydraulic turbine (to transform hydraulic energy to mechanical energy) is determined by the reservoir head, although all types of turbines have similar behavior. Performance is difficult to represent in machine intelligible forms that do not involve equations which, as mentioned above, cannot be efficiently used as indices.

In order to capture performance characteristics, current behavior representations and reasoning mechanisms have to be extended along the following two directions:

- Representation and reasoning about the ranges of operation of a device, and
- Representation and reasoning about the type of variation of behavior variables.

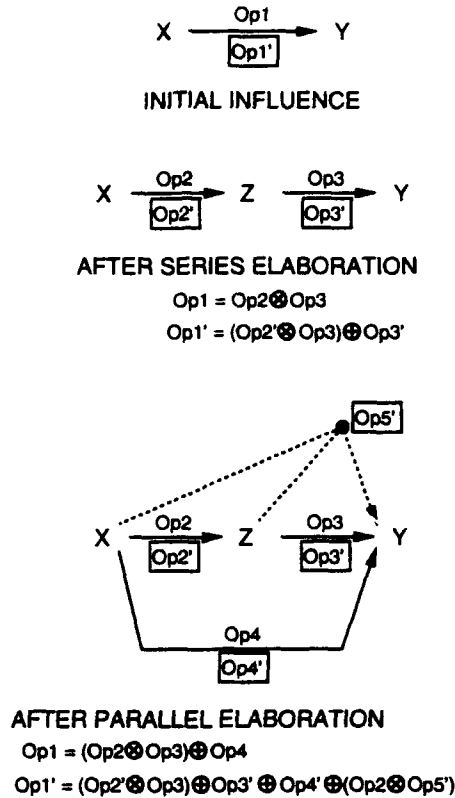


Figure 5-49: Series and parallel elaborations for second-order behavior influence graphs

5.24.1 Ranges of Operation

All devices have an operating range over which they perform with the desired accuracy, and outside of which performance may deteriorate. These ranges include temperature, pressure, humidity, and the ranges of the inputs and outputs of the device. Operating ranges for continuous-behavior devices can be defined by feasible intervals for the inputs and outputs. A collection of similar devices, such as gears of different diameters, has the same ontology as long as the operating range is associated to each member of the family. Given a casebase of devices, retrieval should be done based not only on the influence graph representing the required behavior but also on the ranges of operation of the component or sub-assembly. For instance, the indices for the retrieval of a component could be the influence *Displacement* $\xrightarrow{+}$ *Flow* together with the ranges of operation for the displacement, $[X_1, X_2]$, and for the flow, $[Q_1, Q_2]$. Only components (taps in this case) which match these three indices are retrieved.

Ranges of operation for a design case in a database can be represented by either (a) feasible intervals for inputs and outputs, (b) a mathematical equation that describes the behavior of the component and relates inputs to outputs, (c) an input-output response curve, or (d) experimental data, as illustrated in figure 5-50 for one input and one output. An advantage of using representations (b), (c) or (d) is that the entire ranges need not be matched but only parts of them

as shown by the solid lines. All these four forms of representation can be made parametric so the same case can be retrieved for different ranges of operation, as shown by the second response curve (dashed) in figure 5-50(c)

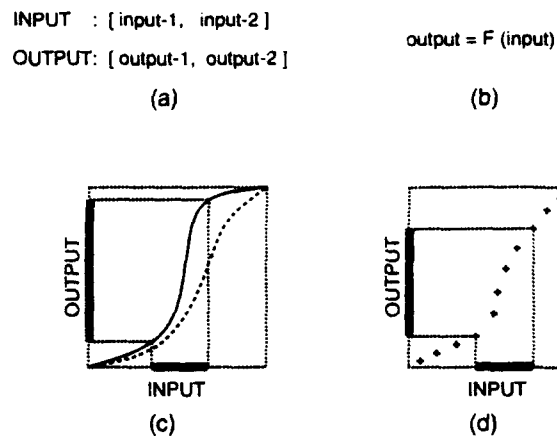


Figure 5-50: Representation of ranges of operation in casebase

For the case where several sub-graphs of an influence graph specification are matched, the ranges of operation have to be propagated from the inputs and outputs towards the other state variables. For example, figure 5-51 shows an influence graph specification (possibly after elaboration of new variables) which is used for retrieval of eight components from a casebase. Influence arcs are not shown for clarity. The behavior influence graph for each component may consist of one or more influences. The ranges of operation (variation) of the inputs and outputs are known and they are used as indices into the casebase for the retrieval of input and output components²³ (components 1, 2, 7 and 8). The ranges of variation of the inputs are propagated *forward* and the ranges of the outputs are propagated *backward* so sets of ranges for the state variables are generated. These sets are then used together with the influence sub-graphs of each component as indices to retrieve intermediate components from the casebase (components 3 to 6). Because the retrieval entails perfect matching of influence graphs and ranges of operation of each component or sub-assembly, the number of feasible designs is greatly reduced in relation to those retrieved by only matching the influence graphs. Relaxation of the ranges of inputs, outputs and/or state variables may be needed when no complete design is retrieved. If such is the case, further tuning of design parameters is necessary so the synthesized device delivers the specified performance.

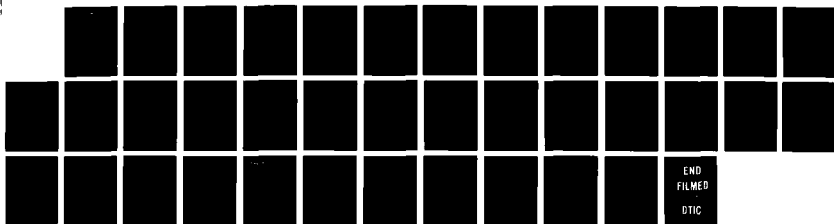
²³An input (output) component is associated with a subgraph of a behavior influence graph that has nodes with in-degree (out-degree) zero

AD-A278 943

CASE BASED REASONING IN ENGINEERING DESIGN(U)
CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST
K SYCARA 30 JUN 93 AFOSR-TR-94-0280 F49620-90-C-0003

UNCLASSIFIED

NL



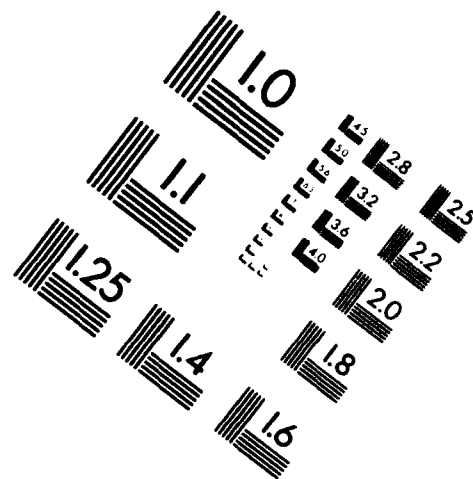
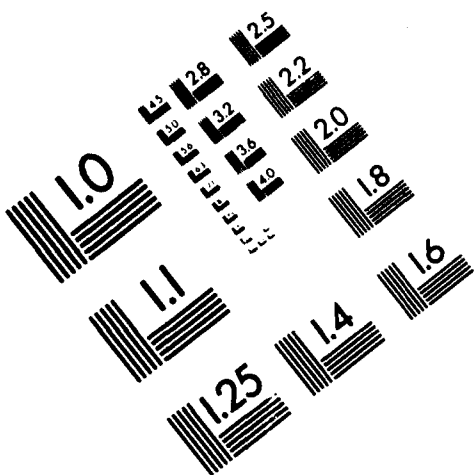


AIM

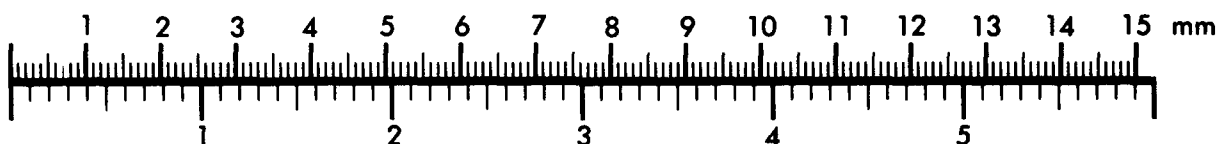
Association for Information and Image Management

1100 Wayne Avenue, Suite 1100
Silver Spring, Maryland 20910

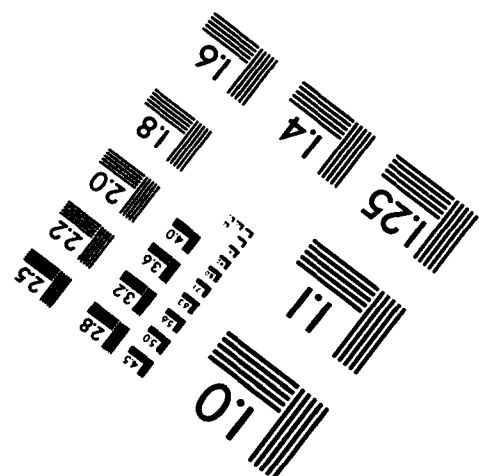
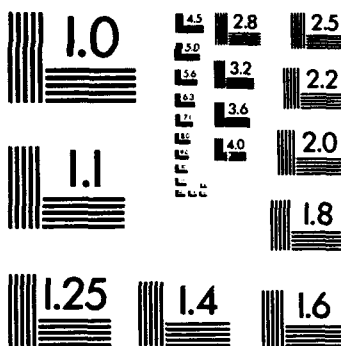
301/587-8202



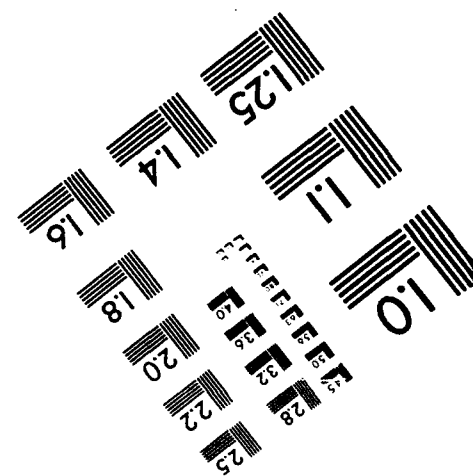
Centimeter



Inches



MANUFACTURED TO AIIM STANDARDS
BY APPLIED IMAGE, INC.



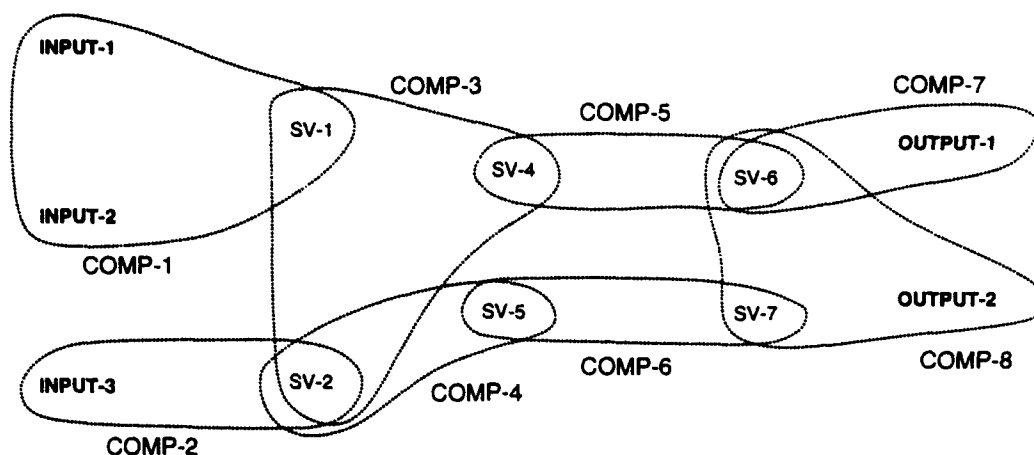


Figure 5-51: Abstraction of influence graph used for retrieval of eight components

5.24.2 Type of Variation of Behavior Variables

Although ranges of operation capture part of the performance information of a design, including the order of magnitude of the behavior variables, they are not sufficient to describe in detail a device response to a given input. That is, the *shape* of a performance curve is not defined by only specifying the ranges of variations of the (dependent and independent) variables.

Characteristics of the response curve can be specified by directly attaching certain information to the qualitative influence graph. For example, if X influences Y positively or negatively, then the simple influence $X \xrightarrow{\pm} Y$ can be replaced by the bi-parametric $X \xrightarrow{(p, q)} Y$ where the parameters p and q capture the *shape* of the dependency relation, as depicted in figure 5-52 after normalizing between 0 and 1. The difference $(q - p)$ is a measure of curvature²⁴ since p and q are defined as the slopes of the response curve at 0 and 1, respectively, and

$$\left[\frac{\partial^2 f}{\partial x^2} \right]_{\text{Mean}} = \int_0^1 \frac{\partial^2 f}{\partial x^2} dx = f'(1) - f'(0) = q - p$$

In order to provide compositionality to the performance representation, an elaboration calculus for performance relations has been developed. For example, if X influences Y according to (p, q) and if an elaboration makes X influence Z that in turn influences Y , then we have to determine the performance relationships between X, Z and Z, Y . The performances should be such that the combined performance preserves the original (p, q) -defined relation between X and Y . The following method seems to be the most appropriate and mathematically sound as shown in the appendix.

The influence $X \xrightarrow{(p, q)} Y$ can be elaborated in series into:

²⁴In fact, it is the mean of the inverse of the curvature

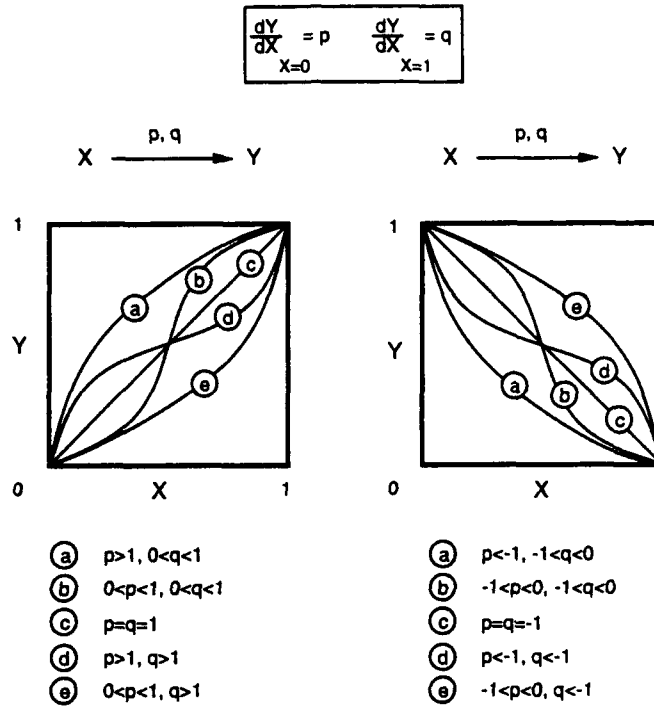


Figure 5-52: Families of Monotonic Influences

$$X \xrightarrow{(p1, q1)} Z \text{ and } Z \xrightarrow{(p2, q2)} Y$$

where: $p1 \cdot p2 = p$ $q1 \cdot q2 = q$ if $p1, q1 > 0$, or $p1 \cdot q2 = p$ $q1 \cdot p2 = q$ if $p1, q1 < 0$
or in parallel into:

$$X \xrightarrow{(p1, q1)} Z, Z \xrightarrow{(p2, q2)} Y \text{ and } X \xrightarrow{(p3, q3)} Y$$

where: $p1 \cdot p2 + p3 = p$ $q1 \cdot q2 + q3 = q$ if $p1, q1 > 0$, or $p1 \cdot q2 + p3 = p$ $q1 \cdot p2 + q3 = q$ if $p1$

Figure 5-53 illustrates the above serial elaboration with operating ranges $[x_1, x_2]$, $[y_1, y_2]$ and $[z_1, z_2]$ for X , Y and Z , respectively. Each performance relation can be used as a primitive and can be assembled to predict the performance of the entire influence graph. Only portions of the ranges of the retrieved components make up the specified operating ranges for X and Y .

Whenever case representation and retrieval is based on the operating ranges of the input and output (as in figure 5-50(a)) and the parameters p and q characterizing the response curve, matching portions of operating ranges entails the parametric approximation of the response function. For x_N and y_N normalized between 0 and 1, an approximated response function is given by the following cubic polynomial.

$$y_N = \Phi(x_N) = (p + q - 2) x_N^3 + (3 - 2p - q) x_N^2 + p x_N$$

Given the specification influence $X \xrightarrow{[\alpha, \beta]} (\pi, \theta) \rightarrow Y \xrightarrow{[\chi, \delta]}$ where $[\alpha, \beta]$ and $[\chi, \delta]$ denote the operating ranges for X and Y , then case $X \xrightarrow{[a, b]} (p, q) \rightarrow Y \xrightarrow{[c, d]}$ is retrieved if and only if either

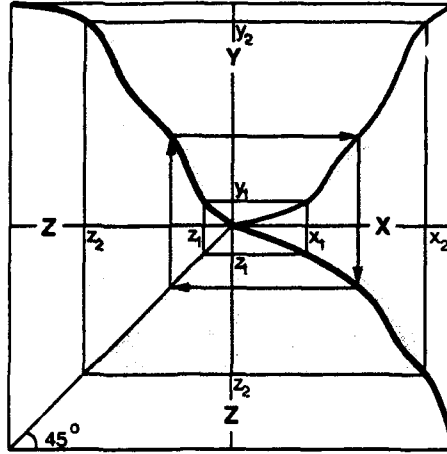


Figure 5-53: Composition of Performance Characteristics

- $[\alpha, \beta] \approx [a, b]$, $[\chi, \delta] \approx [c, d]$, $\pi \approx p$, and $\theta \approx q$, as depicted in figure 5-54(a), or
- $[\alpha, \beta] \subseteq [a, b]$, $[\chi, \delta] \subseteq [c, d]$, $\chi_N \approx \Phi(\alpha_N)$, $\delta_N \approx \Phi(\beta_N)$, $\pi \approx \Phi'(\alpha_N)$, $\theta \approx \Phi'(\beta_N)$, as depicted in figure 5-54(b), where the normalized parameters are defined as follows

$$\alpha_N = \frac{\alpha - a}{b - a} \quad \beta_N = \frac{\beta - a}{b - a} \quad \chi_N = \frac{\chi - c}{d - c} \quad \delta_N = \frac{\delta - c}{d - c}$$

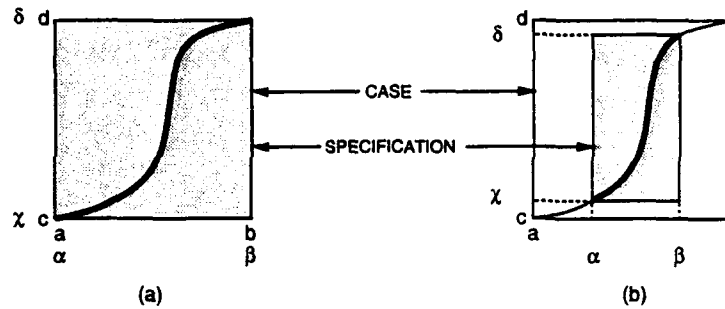


Figure 5-54: (a) Complete and (b) partial matching of case by specification

Approximate equality " $\approx$ " has been used to emphasize that perfect matching between case and specification is not required but only an adequate approximation. However, this may require tuning of design parameters and verification that the synthesized device performs according to specification. The level of accuracy for the matching has to be specified beforehand.

Appendix: Proof of Bi-Parametric Influence Elaborations

Proofs of series and parallel elaborations are based on an assumption of continuity, differentiability and integrability.

Series Elaboration

Let f and g be monotonic functions with domain and range $[0, 1]$. Let h be the composed

function of f and g , i.e., $h = f \circ g$, so its range and domain is also $[0, 1]$.

The measures of curvature for f , g and h are defined as follows

$$\left[\frac{\partial^2 f}{\partial x^2}\right]_{Mean} = \int_0^1 \frac{\partial^2 f}{\partial x^2} dx = f'(1) - f'(0) = q_f - p_f$$

$$\left[\frac{\partial^2 g}{\partial x^2}\right]_{Mean} = \int_0^1 \frac{\partial^2 g}{\partial x^2} dx = g'(1) - g'(0) = q_g - p_g$$

$$\left[\frac{\partial^2 h}{\partial x^2}\right]_{Mean} = \int_0^1 \frac{\partial^2 h}{\partial x^2} dx = h'(1) - h'(0) = q_h - p_h$$

But

$$\begin{aligned} h'(1) - h'(0) &= f'(g(1)) \cdot g'(1) - f'(g(0)) \cdot g'(0) \\ &= f'(1) \cdot g'(1) - f'(0) \cdot g'(0) \quad \text{if } g'([0, 1]) > 0 \\ &= f'(0) \cdot g'(1) - f'(1) \cdot g'(0) \quad \text{if } g'([0, 1]) < 0 \end{aligned}$$

Therefore

$$\begin{aligned} q_h - p_h &= q_f \cdot q_g - p_f \cdot p_g \quad \text{if } g'([0, 1]) > 0 \\ &= p_f \cdot q_g - q_f \cdot p_g \quad \text{if } g'([0, 1]) < 0 \end{aligned}$$

Selection of q_f , q_g , p_f and p_g such that $q_h = q_f \cdot q_g$ and $p_h = p_f \cdot p_g$ if g is an increasing function, or $q_h = p_f \cdot q_g$ and $p_h = q_f \cdot p_g$ if g is a decreasing function is enough to satisfy the above conditions.

Parallel Elaboration

Since multiplication of parameters has been proved for influence in series, it is sufficient to prove the additivity of parameters when the influences are in parallel.

Let f be a monotonic function with domain $[0, 1] \times [0, 1]$ and range $[0, 1]$. We have that

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial^2 f}{\left[\frac{\partial x_1}{\partial x}\right]^2} + \frac{\partial^2 f}{\left[\frac{\partial x_2}{\partial x}\right]^2} + \frac{\partial f}{\partial x_1} \frac{\partial^2 x_1}{\partial x^2} + \frac{\partial f}{\partial x_2} \frac{\partial^2 x_2}{\partial x^2}$$

Setting x_1 and x_2 equal to x and taking the mean of the second derivative over $[0, 1]$, we obtain

$$\left[\frac{\partial^2 f}{\partial x^2}\right]_{Mean} = []_{Mean} + []_{Mean}$$

which proves the additivity of parameters for influences in parallel.

Chapter 6

CARD: The CADET casebase

"An excellent technique for utilizing this compilation of laws and effects in problem solution is to break the problem into input and output physical quantities. As a simple example, you may want a light signal input and a temperature variation output. First, try bridging the gap directly with any known effect. If this is unsuccessful, think of a number of physical quantities that are suggestive as intermediate steps. Light may be converted to resistance, to current, to heat. These steps may be connected with as many different phenomena as can be recalled from memory. A number of semi-complete paths will result, each with a different link missing. A renewed search of unfamiliar laws and effects is certain to complete one or more of these links.

Hix and Alley, Physical Laws and Effects, John Wiley & Sons, 1959

6.25 Introduction

Design is the activity of generating a description of an artifact that meets a set of functional specifications. The design process is generally classified into conceptual design, parametric design and detailed design. In the conceptual design phase, the designer explores different choices of physical effects, geometry and materials to create a set of feasible designs that could potentially meet the required specifications. During parametric design, the relations between the different design parameters of the artifact are established and preliminary analysis is performed to specify values for the parameters. In detailed design, the actual detailing of the artifact is done with all the specifications outlined. During all these phases, the designer uses his knowledge of the domain, experiential knowledge gained through previous designs and information from various disparate sources to aid in the design process. Design is partly adaptive and partly routine. In adaptive design, old designs are retrieved and adapted for a new functionality.

The basic tenet of this effort is that placing at a designers fingertips the capability to access a repository of prior designs thorough a variety of indexing schemes can aid a designer in novel and routine design.

Experiential knowledge plays a key role in the design activity and can be aided by using "analogical" inference mechanisms. Designs, if properly indexed and archived, can be retrieved based on a set of critical "keys". Case-based reasoning (CBR) is a problem solving

paradigm where previous experiences are used to guide problem solving [5, 27, sycara.negotiation.87, 18]. Cases similar to the current problem are retrieved from memory, the best case is identified and compared with the given problem. The case is adapted to fit the current situation based on identified differences between the precedent and the current case. Successful cases are archived to be reused in the future. Failed cases are archived to warn the problem solver of potential failures and help recover from failures. CBR is relevant in domains where a case can be clearly identified by an expert, cases can be compared and inferences drawn to solve a new problem, cases can be generalized, and cases retain their utility over long periods of time.

Engineering design is a domain in which a case-based approach could be very useful [71].

Research in case-based design deals with identifying relationships between artifact behavior and structure, developing representations that can capture this behavior, inference strategies for design, indexing schemes to identify key characteristics in tightly-coupled components and strategies to use cases at different levels of abstraction. CADET is a tool that aids the design process by retrieving relevant prior designs from a database of designs. The retrieved designs could then be assembled to meet the functionality required. CARD (Case Retrieval for Design) which is the elaboration and retrieval module of CADET has been implemented using the case-based reasoning paradigm for storing and retrieving designs. Designs have been indexed using "influence graphs" a variant of qualitative algebra [68, 46]. CADET is a prototype case-based design tool for design of electro-mechanical artifacts.

6.26 Description of CADET

CADET is a system that performs conceptual design by synthesizing a device from snippets accessed from previous design cases. CADET uses a multi-layered representation to express function, behavior, structure and related constraints. CADET's approach to design has three main characteristics, (a) transformation of high level functional specifications to a set of alternative design descriptions that satisfy the given specifications, (b) adaptation of retrieved previous designs and design pieces to fit specifications and constraints in the current design problem and (c) synthesis of the most promising alternative(s) from components whose combined behavior is equivalent to the overall device specifications. The input to CARD is an "influence graph" that specifies the required behavior of a device and a set of physical constraints. CARD has access to a case-memory and engineering domain laws and principles.

The system based on the input behavioral specifications performs transformations of behavioral indices and uses these indices to retrieve and match cases from the case memory. Details regarding the behavioral transformations are presented in [47]. CARD has been implemented in COMMON LISP. To deal with issues of scalability, cases i.e. previous designs are stored in the Informix relational database. The current implementation supports a LISP-SQL interface and is used to extract data from the database as and when required during the design process. The overall system architecture showing CADET and CARD is in Figure 6-55.

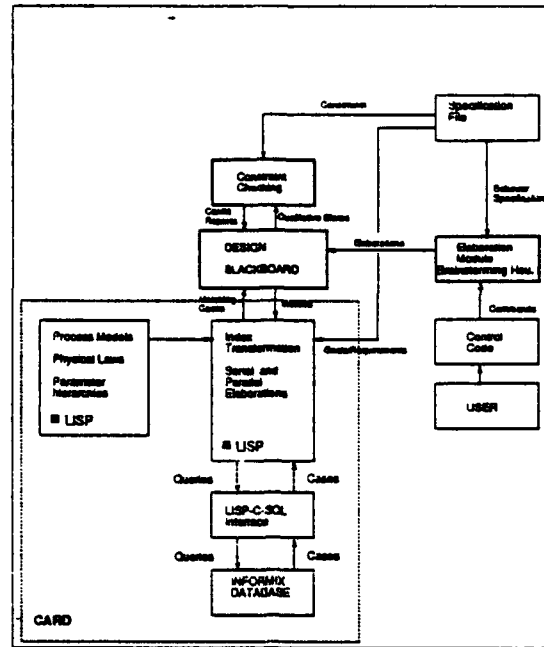


Figure 6-55: CADET system architecture

6.26.1 Case representation

A case-base has to be indexed properly so as to aid the retrieval process. Cases in CARD could be indexed based on a number of abstractions. The abstractions that could be used are:

- Linguistic descriptions.
- Behavioral descriptions through the use of *influence graphs*. Qualitative states.
- Pictorial descriptions through sketches, solid models and schematic diagrams.
- Structural features including material, shape and device topology.
- Manufacturing descriptions through use of process plans and assembly plans.
- Performance attributes such as cost, reliability, availability, material, and weight.

In this implementation, the indices for cases are *influence graphs*. In CARD, device behavior is represented as a collection of influences among various inputs and outputs. An influence is a qualitative differential (partial or total) relation between two variables one of which is a dependent variable and the other an independent variable. Influences are organized as graphs. An influence graph is a directed graph whose nodes represent the variables of concern and whose arcs are qualitative values (+,-,0) depicting the influence of one variable on another. The influence sign denotes the monotonicity the first order differential between the two variables of interest. These graphs of influences are used to represent the behavior of devices, where each influence corresponds to some physical effect.

Consider, for example, a Pascal press which is hydraulic device for pressure amplification. Figure 6-56 shows a two and a half dimensional representation of the press. Input pressure is P1 and the output pressure is P2. As P1 increases, P2 increases or when P1 decreases, P2 decreases. This represented as $P1 \{-(+)\rightarrow\} P2$ which is an influence meaning that the output pressure

increases monotonically with the input pressure. The ratio of the pressures is equivalent to A_1/A_2 but is not necessary to denote the relation between the pressures. In the LISP representation an influence is denoted as a list $\{(P_1 P_2 +)\}$ i.e. { (left-parameter right-parameter sign)}. An influence graph is a list of influences. Consider a cascade of Pascal presses (see Figure 6-56). P_1 is the input and P_3 is the output. The influence graph is denoted as $((P_1 P_2 +) (P_2 P_3 +))$.

(see next page)

Figure 6-56: A Pascal press

Cases are represented in LISP as follows:

```
;;;Defstruct definition for a case
(defstruct case
  device_name ;symbol
  comp_name   ;list of components
  infl_list   ;list of list of influences (((left right sign) (1
                                     ;right sign)). Each element in the list correspond
                                     ;a component's influences

  device_type ; Name
  prop_list   ; List of properties of the device
)
```

Files containing such structures can be created using an editor and saved. For example, the following is a file *case.lisp* containing two cases.

```
(cs1 cs2)
#s(CASE DEVICE_NAME SPRING-2 COMP_NAME (SPRING-2) INFL_LIST
  (((FORCE-1 TRANSLATION-DISPLACEMENT-1 +)
   (FORCE-1 ELASTIC_ENERGY-1 -)))
  DEVICE_TYPE SPRING
  PROP_LIST ((20 10 BRASS 10 20 30 30 10 70 10)))
```

```
#s(CASE DEVICE_NAME SPRING-4 COMP_NAME (SPRING-4) INFL_LIST
  (( (TRANSLATION-DISPLACEMENT-1 FORCE-1 -)
    (TRANSLATION-DISPLACEMENT-1 ELASTIC_ENERGY-1 -)))
  DEVICE_TYPE SPRING
  PROP_LIST ((10 30 STEEL 10 20 30 30 20 20 20)))
```

The first line in the file is a list of symbols identifying the case structures. The first structure refers to a spring denoted by *DEVICE_NAME*. It has one component, the spring itself denoted by *COMP_NAME*. If there are multiple components, they should be listed together. The *INFL_LIST* slot refers to the influence lists for each of the components. For example, consider a case with two components. (*COMP_NAME* (a b)). Then *INFL_LIST* is ((*list of influences of component a*) (*list of influences of component b*)). In the case of the spring there is only one component and its list of influences (the influence graph) is listed. The slot *DEVICE_TYPE* denotes the domain of the device such as electrical, hydraulic, mechanical and magnetic and is the general device name. The *PROP_LIST* slot is a list of property values. The properties considered are cost, weight, type of material, qualitative cost, qualitative weight, availability, demand, reliability, adaptability and performance. The property definitions are presented below.

These structures can be loaded into the lisp environment by using the functions in the file *case_input.l*. These structures are normalized and then loaded into a relational database. *Normalization* is a relational database operation by which many-to-one and one-to-many relationships among entities are converted to one-to-one relationships. On normalization of influences, in the case of the spring two lists result.

- (*SPRING-2 SPRING-2 FORCE-1 TRANSLATION-DISPLACEMENT-1 +*) (*SPRING-2 SPRING-2 FORCE-1 ELASTIC_ENERGY-1 -*)

The one-to-many relationship between a component and its influences is resolved into one-to-one relationships. Each of the above list then forms a *tuple* in a relational database.

The database consists of two tables, *Influences* and *Properties*. The *Influences* table has the following columns:

- *DEVICE NAME* -- Name of the device
- COMPONENT NAME* -- Component of the above device
- LEFT-INFLUENCE* -- Left parameter in an influence. This is the *input* parameter.
- RIGHT-INFLUENCE* -- Right parameter in an influence. This is the *output* parameter.
- SIGN* -- Sign of monotonicity relationship between parameters (+, - or 0)

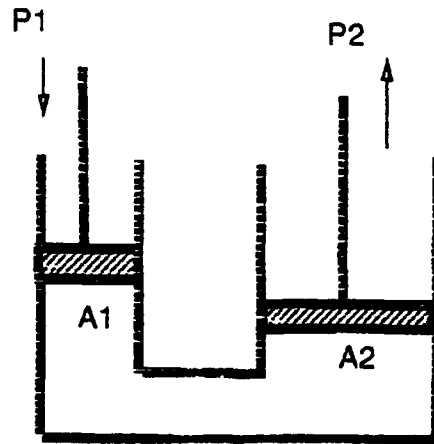
Each tuple is a horizontal row in the table. See Figure 6-57.

Similarly, on normalization, the *PROP_LIST* slot yields the following list for the spring, *SPRING-2*:

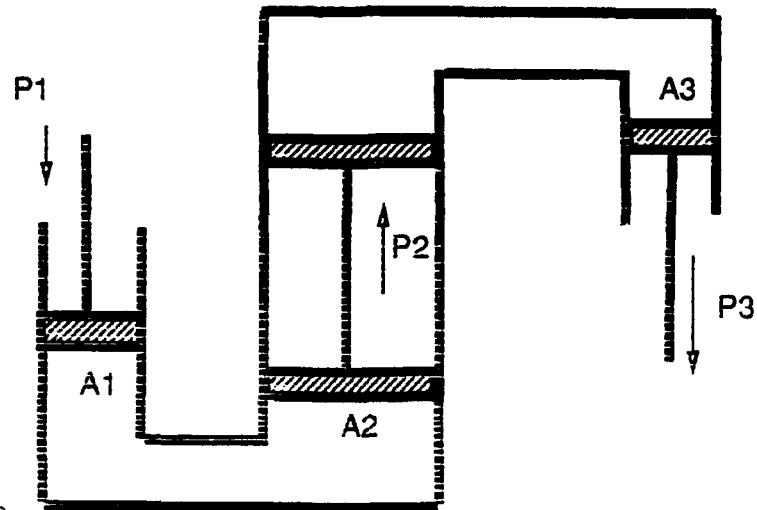
- (*SPRING-2 SPRING-2 SPRING 10 20 BRASS 10 20 30 30 10 70 10*)

If a device has multiple-components, each component would yield one list on normalization. The above list (a tuple) is then an entry into the *Properties* table. The *Properties* table has the

PASCAL PRESS



$$P_1 \xrightarrow{+} P_2$$



A CASCADE OF PASCAL PRESSES

$$P_1 \xrightarrow{+} P_2 \xrightarrow{+} P_3$$

TABLE OF INFLUENCES				
DEVICE_NAME	COMP_NAME	LEFT_INF	RIGHT_INF	SIGN
SPRING-2	SPRING-2	FORCE-1	ENERGY-1	-
SPRING-2	SPRING-2	FORCE-1	T-DISP-1	+

TABLE OF PROPERTIES					
DEVICE_NAME	COMP_NAME	DEV_TYPE	COST	WEIGHT	
SPRING-2	SPRING-2	SPRING	10	20	

Figure 6-57: Relational database tables

following columns:

- **DEVICE NAME** -- Name of device (e.g. *SPRING-2*)
- **COMPONENT NAME** -- Name of component (e.g. *SPRING-2*)
- **DEVICE TYPE** -- Electrical, hydraulic, mechanical, magnetic (e.g. *SPRING*)
- **COST** -- Cost of the component (e.g. *10*)
- **WEIGHT** -- Weight of the component (e.g. *20*)
- **MATERIAL** -- Material of the component (e.g. *BRASS*)
- **QUALITATIVE COST** -- A ranking between 0 and 100 (0 is low cost) (e.g. *10*)
- **QUALITATIVE WEIGHT** -- A ranking between 0 and 100 (0 is low weight) (e.g. *20*)
- **AVAILABILITY** -- A ranking between 0 and 100 (0 is high availability) (e.g. *30*)
- **DEMAND** -- A ranking between 0 and 100 (0 is low demand) (e.g. *30*)
- **RELIABILITY** -- A ranking between 0 and 100 (0 is low reliability) (e.g. *10*)
- **ADAPTABILITY** -- A ranking between 0 and 100 (0 is low adaptability) (e.g. *70*)
- **PERFORMANCE** -- A ranking between 0 and 100 (0 is low performance) (e.g. *10*)

In the list generated on normalization, the ordering of the elements corresponds to the order of properties listed above.

Each case is a device and is assumed to be an assembly of components with a specific behavior. If the component is part of a device, only a part of those influences might be relevant for behavior in that device and only those influences should be listed in the database. Different devices can be made with piece-meal selections of components and assembling them.

Components and sub-assemblies form snippets of a case while a device is a complete case by itself.

The vocabulary that describes the types of parameters that arise in the influences is as follows:

- FLUIDIC
 - FLOW_VOLUME
 - FLOW_RATE
 - FLOW_RATE-LIQUID
 - FLOW_RATE-LIQUID-WATER
 - FLOW_RATE-LIQUID-OIL
 - FLOW_RATE-GAS
 - FLOW_RATE-GAS-AIR
 - PRESSURE
 - PRESSURE-LIQUID
 - PRESSURE-GAS
 - PRESSURE_MOMENTUM
- KINEMATIC
 -
 - TRANSLATION
 - TRANSLATION-DISPLACEMENT
 - TRANSLATION-VELOCITY
 - TRANSLATION-ACCELERATION
 - ROTATION
 - ROTATION-DISPLACEMENT
 - ROTATION-VELOCITY
 - ROTATION-ACCELERATION
- KINETIC
 -
 - FORCE
 - LINEAR_MOMENTUM
 - TORQUE
 - ANGULAR_MOMENTUM
 - STRESS
- THERMAL

- - TEMPERATURE
 - ENTROPY_FLOW
 - HEAT_ENERGY
 - HEAT_FLUX
- FORM
- - LENGTH
 - AREA
 - VOLUME
 - ANGLE
 - MASS
 - MOMENT_OF_INERTIA
- ELECTRIC_MAGNETIC
- - VOLTAGE
 - CURRENT
 - CHARGE
 - FLUX_LINKAGE
 - ELECTRIC_ENERGY
 - MAGNETIC_ENERGY
 - ELECTRIC_POWER
 - MAGNETIC_FLUX_DENSITY
 - CAPACITANCE
 - RESISTANCE
 - INDUCTANCE
- MECHANICAL ENERGY
- - GRAVITATIONAL_ENERGY
 - KINETIC_ENERGY
 - ELASTIC_ENERGY
 - MECHANICAL_POWER
- TEMPORAL

-
- TIME
- FREQUENCY
- PERIOD

Parameters are named according to the following convention:

- For any parameter append a suffix integer. For example *ANGLE-1* and *ANGLE-2* are two parameters.
- Make sure that two parameters that mean different physical values do not have the same name.
- Unknown variables instantiated by the system during elaboration of a specification are represented by *UNKN-1*, *UNKN-2*, etc.

The present number of cases is 125. The relational database that stores the case is defined by the variable **database** in the system. Also the shell environment variable *DBPATH* must be set to the path where the database is located. Cases were obtained through a number of sources. A summary list of the sources is as follows:

- *The Way Things Work*.
- Commercial catalogs.
- Artobolevsky's *Mechanisms in Modern Engineering Design*.
- Grafstein's *Pictorial Handbook of Technical Devices*.

They are in the sub-directory *textdb*. The domains they cover are:

- displacement transducers
- springs
- force and torque transducers
- steam turbines
- hydromechanical
- temperature transducers
- machine elements
- transformers
- pressure regulators
- velocity and acceleration transducers
- pressure transducers

6.26.2 Case Retrieval

Input specifications to CARD is an influence graph describing the required behavior. Cases are retrieved from the case base if valid solutions are available otherwise a process of "elaboration" is carried out and a set of new influence graphs generated. Elaboration [47] is of two types, serial and parallel elaboration and follow rules of qualitative algebra. Each elaboration introduces an unknown variable in the initial graph. It is represented by a symbol like *UNKN-1* where the suffix integer identifies the number of unknown variable. Both serial and parallel elaboration, generate a set of influence graphs with a different topology than the initial influence graph. The elaboration process is explained further in later sections.

6.27 Getting Started

6.27.1 Sources

CARD1.0 is intended to be completely portable and should run under any Common Lisp implementation. CARD1.0 can be obtained through anonymous ftp from *ftp.cs.cmu.edu*. Change directory to */afs/cs/project/cadet/ftp*. The file *card1.0.tar.Z* can be retrieved. After retrieval, type *zcat card1.0.tar.Z | tar xvf -* to recreate the sub-directory structure. Edit path variables in *card.system* to reflect the new directory structure.

6.27.2 Setting up the database

A relational database such as INFORMIX is required²⁵. Two tables *influences* and *properties* tables must be created in these databases. The tables can be created using the *createdb* SQL command or the 4GL interface provided with these databases. The columns for the tables are described in Section 4. After creating the tables, set the shell variable *DBPATH* to the directory where the database is located. Edit the *cadet.system* file and set **database** to the name of the database. Also set **MAIN-PATH** to the path where the CARD files are located. The file *sql/c_sql.ec* needs to be compiled using the *libsql.a*. For Informix systems, it can be compiled using *esql -c -G 0 c_sql.ec*. A file *c_sql.o*. If there are problems in compilation for other database systems, some of the basic Embedded SQL function call names might be different and have to be changed accordingly in the *.ec* files and recompiled. If no database system is available, the cases can be stored as structures in the LISP environment. A manual that describes the LISP-SQL interface between Informix and Allegro CL²⁶ is available in the *doc* sub-directory. A minimal set of LISP functions that provide access to the RDBMS is described. C functions that contain embedded SQL functions are compiled into object code and accessed from LISP through the foreign function interface. Different databases and LISPs might have different foreign function implementations and associated libraries. Setting up the interface for an

²⁵INFORMIX and RDSQL is a registered trademark of Relational Database Systems, Inc

²⁶Allegro Common Lisp is a trademark of Franz Inc.

INFORMIX DBMS is described in the following section. The retrieval function that matches the influences in the case structures is to be coded and called from the search functions. The search and retrieval functions are described in later sections. Details regarding bugs, suggestions can be sent to madhu@ri.cmu.edu. CARD is available free of charge and without any restriction. Please send mail to the above address if you obtain a copy anyway so that we may keep track of who has obtained a copy and keep users informed of enhancements and bug-fixes by the way of a mailing list.

6.27.3 Setting up the system

The location of the foreign function libraries for the Common Lisp being used need to be identified and the file *cadet.system* file has to be edited. Foreign function libraries that are required with an INFORMIX database are:

- *libsql.a*
- *libutil.a*
- *librds.a*
- *libc_g.a*

The above libraries might be different for other databases. The module *foreign1* needs to be edited in *card.system*. Start up lisp (*cl, lisp*) and load the file *defsystem.lisp*. Load *card.system* (after editing it to reflect the above mentioned changes).

6.27.4 Using the system

Assuming that the database has been created and the necessary tables created, the following steps have to be followed to load and query the database:

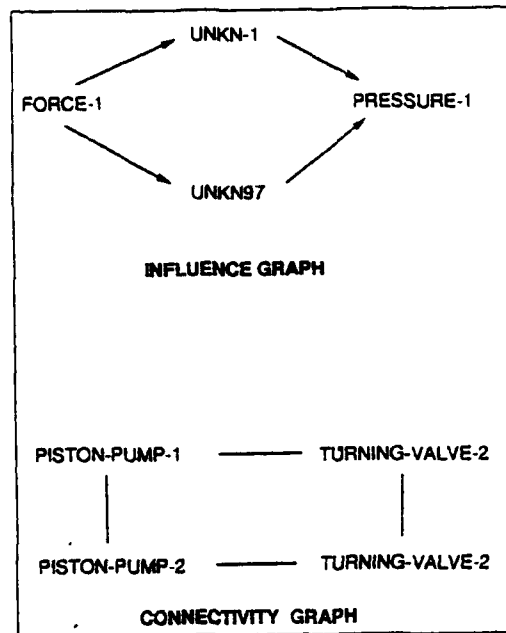
1. Startup common lisp.
2. Load the file *defsystem.lisp*.
3. Load the file *card.system*.
4. To load all files, type (*card*) at the LISP prompt. Respond accordingly to the the prompts, the system will load all the files. At this stage you might get errors if the library files are not available.
5. To load the database:

```
;To load all the cases into the database defined in *da
;Assuming the relational database has been setup.
>(create-casebase *database*)
;This function is in the file interface/ui.1
;If no database has been set up, the case structure fil
;loaded directly
```

6. Once the casebase is created, the system can be queried. The input to the system is an influence graph. Let us say one is looking for a device that causes a pressure variation with force. %Also you want to use an intermediate variable.

7. >(setq inf '((FORCE-1 UNKN1 +) (UNKN1 PRESSURE-1 -)))
8. You have set *inf* to the behavior specification. To elaborate and retrieve cases, type
9. >(elabsearch inf)
10. The following is a trace as the system searches for a set of cases that match the specification.
11. **Elaboration and Retrieval search begun....**
 Retrieval search for ((FORCE-1 UNKN1 +) (UNK
 Expanding Retrieval node SNODE0
 Influence is (FORCE-1 UNKN1 +)
 Number of new retrieval child nodes is
 Expanding Retrieval node SNODE1
 Influence is (TRANSLATION-DISPLACEMENT
 Number of new retrieval child nodes is
12. The output of the system is as follows:
13. **Elaborated influence is ((FORCE-1 UNKN1 +) (FORCE-1 UNK
 (UNKN97 PRESSURE-1 +) (UNKN1 PRESSU
 Alternative solutions (sets of cases) for the above in
 ((PISTON-PUMP-1 PISTON-PUMP-2 TURNING-VALVE-2 TURNING-
 (PISTON-PUMP-1 PISTON-PUMP-2 TURNING-VALVE-2 SLIDING-
 (PISTON-PUMP-1 PISTON-PUMP-2 PISTON-PUMP-1 TURNING-VA
 (PISTON-PUMP-1 PISTON-PUMP-2 SLIDING-VALVE-1 TURNING-
 (PISTON-PUMP-1 PISTON-PUMP-2 SLIDING-VALVE-1 SLIDING-
 (PISTON-PUMP-1 PISTON-PUMP-2 PISTON-PUMP-1 SLIDING-VA**
14. The initial specification has been elaborated and a set of matching cases are found. In this example six solutions that match the influence graph have been found. Each set of cases is then presented. Consider the solution (*PISTON-PUMP-1 PISTON-PUMP-2 TURNING-VALVE-2 TURNING-VALVE-2*). The first device *PISTON-PUMP-1* corresponds to the first influence (*FORCE-1 UNKN1 +*), the second device to the second influence and so on. The output is a connectivity graph denoting the way the devices should be assembled together based on the topology of the corresponding influence graph. In Figure 6-58, the graph on top is the topology of the elaborated influence graph. For each arc in this graph, there is a corresponding device in the graph shown at the bottom of the figure. The connectivity graph is a dual of the influence graph. The device *TURNING-VALVE-2* has been retrieved twice for two different influences.
- 15.

The next version shall include an interface to view the influence graph and connectivity graph. At present this interpretation has to be performed manually. Elaboration and retrieval processes are interleaved to search for feasible solutions. The file *EXAMPLES* contains examples of the functions and the options that can be used with the system. Also a number of additional examples showing the influence graph and the output are in the file *EXAMPLES*. One can generate multiple number of solutions by defining the number of elaborations required and the



16. **Figure 6-58:** Influence and Connectivity Graphs

number of cases required for each elaboration.

6.28 Bugs present and Fixes

CARD is a research prototype and might have conceptual bugs as well as programming errors. %Bugs that have been identified so far are: Run the examples. The test functions cover all the functions in the code and return the right results. Errors in creating the case-base files, data-base retrieval errors, memory-problems in the LISP-SQL interface are highly probable. It is probable that a segmentation error (memory violation) occurs when the query functions interact with the database. The system will hang and needs to be restarted. This may happen due to non-availability of free memory from the stack.

Common errors committed by users are:

- Not enclosing influences in a list like *(inf1 inf2..)* where *inf* is of the form *(a b +)*.
- Errors also could occur when a case is not represented properly in the case base. Incomplete influence lists and bad parenthesis locations are potential bugs.

Please be careful to avoid them. For clarifications, bug-fixes and suggestions, send E-mail to madhu@cs.cmu.edu.

6.29 Index transformation and Retrieval

The input to the system is a behavior that is desired and is represented as an influence graph as shown in Section 2. The case-based design system works through a process where elaboration and retrieval are interleaved. The main algorithm consists of two search algorithms described by ELAB_SEARCH and RETRIEVAL_SEARCH. Both these algorithms take an influence graph as an input.


```

orig-left      ;;original left match
right          ;;case right
left           ;;case left
sign           ;;case sign
device-name    ;;device-name
comp-name      ;;comp-name
rank           ;;Evaluation function va
synthesis-measure ;;synthesis measure..a h
               ;;allocated dynamically
perf_chars     ;;a list of performance
               ;;component.;; list as c
)
;;;Stores the remaining influence graph to be matched w
instantiated parameters.
(defstruct match_info      ;;Symbol is MI
  inf_graph                ;;The inf-graph remaining to b
)

```

3. Consider the first influence of GRAPH and call it INDEX. Call the list of remaining influences REMAINING-GRAPH. Set the *orig-right*, *orig-left* and *sign* slots of the *match* structure to the right parameter, left parameter and sign of INDEX. Set the *inf_graph* slot of *match_info* structure to REMAINING-GRAPH. Put the node in OPEN.
4. If OPEN is nil, stop and return CASES. Select first element of OPEN. Call it RET-NODE.
5. If the *inf_graph* slot of the *match_info* structure of RET-NODE is nil, then stop. Return CASES.
6. Set INDEX to influence created from the *orig-right*, *orig-left* and *sign* slots of the *match* structure of RET-NODE.
7. Set REMAINING-INDEX to *inf_graph* slot of *match_info* structure of RET-NODE. Then do
 - If INDEX is nil, trace path from the root of the tree to RET-NODE and return the list of cases and associated influence created from *right*, *left* and *sign* slots of *match* structure in CASES. Search database to retrieve all instances of INDEX in different components of different cases in the database.
 - There are three possible forms for INDEX:
 - Both left parameter and right parameter are known. Then cases that have that particular influence with the required sign are retrieved.
 - Left parameter is unknown (specified as *UNKN-1*, *UNKN-2*, etc. in the influence). Then cases that match the right parameter and the sign are retrieved. Right parameter is unknown (specified as *UNKN-1*, *UNKN-2*, etc. in the influence). Then cases that match the left parameter and the sign are retrieved.
 - Each retrieved case is a list consisting of five elements which are the device

name, component name, the left parameter, the right parameter and the sign i.e. all the tuples containing the influence are retrieved. Also retrieved is the properties of each component in which the influence occurs and the *perf_chars* slot of *match* structure is set.

- If INDEX has no unknowns, associate the names of all the matched cases with INDEX. Do
 -
 - If there are unknown parameters in INDEX, find all influences in the retrieved cases that match the known part of the influence. Call these influences MATCHINGS.
 - For each influence (X) in MATCHINGS, create a copy of REMAINING-INDEX, replacing all occurrences of the unknown variable in INDEX, by the corresponding variable in X.
 - Associate with each X, the case that contains the influence and the corresponding REMAINING-INDEX.
 - Create new nodes for each of the matched cases and the associated REMAINING-INDEX. The *match* structure takes the first influence of REMAINING-INDEX. The *match_info* structure takes the rest of REMAINING-INDEX.
 - Store them as child nodes of RET-NODE. Put a list of these nodes in OPEN.
8. Sort OPEN in the increasing order of the *rank* slot of the *match* structure of each node. The rank is calculated through an evaluation function on the properties of the case that is retrieved from the case-base. Go to Step 2.

6.29.1 Elaboration tree

Algorithm ELAB_SEARCH creates a *elaboration tree*. The elaboration tree is persistent till the final solution for the input graph is found. The search can also be stopped by setting a limit on the depth of the elaboration tree through the parameter **elab-tree-depth**. The parallel and serial elaboration process create a large number of children nodes. If the initial influence graph consists of two influences, then on elaboration eight child nodes would be created, two through serial elaboration and six through parallel elaboration. See Figures 6-59,6-60. In Figure 6-60 the new influences have negative signs and the qualitative relationship between *a,b* and *c* is not violated.

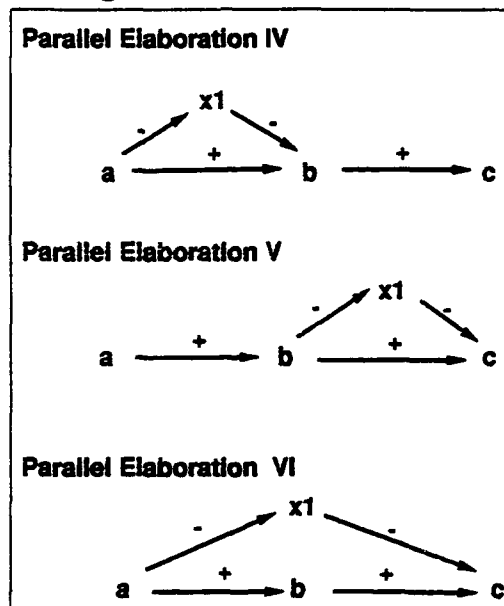
The elaboration tree is as in Figure 6-61.

The branching factor of the elaboration tree grows with each level of elaboration exponentially as new variables are introduced at each level.

As one can see, the number of nodes grows very fast. So retaining a small search space is essential. Two rules are incorporated in the system to reduce the search space of the elaboration tree. The rules are:

1. In an influence graph of the form $((a\ x1\ +)\ (x1\ x2\ +)\ (x3\ b\ +))$ where *a* and *b* are

(see next page)

Figure 6-59: Elaborations-I**Figure 6-60: Elaborations-II**

parameters and $x1, x2, x3$ are unknowns; all the influences containing the input parameters ($a \ x1 \ +$) and output parameters ($x3 \ b \ +$) must be matched with cases in the case-base using RETR_SEARCH. If for any one of those influences no matches can be found, it is futile to elaborate and match any further as no solutions can be found for any new elaborations.

2. A parallel elaboration will not lead to any successful retrievals if the serial elaboration that is embedded in the parallel elaboration with one less unknown variable is unsuccessful. For example if the serial elaboration $((a \ x1 \ +) (x1 \ b \ +))$ is unsuccessful, the parallel elaborations $((a \ x1 \ +)(a \ x2 \ +) (x2 \ b \ +) (x1 \ b \ +))$ and $((a \ x1 \ +)(a \ x2 \ +) (x2 \ x1 \ +) (x1 \ b \ +))$ and $((a \ x1 \ +)(x1 \ x2 \ +) (x2 \ b \ +) (x1 \ b \ +))$ will all be unsuccessful.

Original Influence Graph



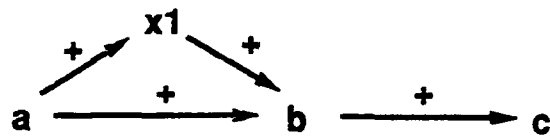
Serial Elaboration I



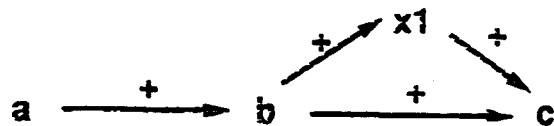
Serial Elaboration II



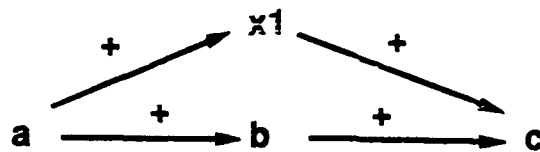
Parallel Elaboration I



Parallel Elaboration II



Parallel Elaboration III



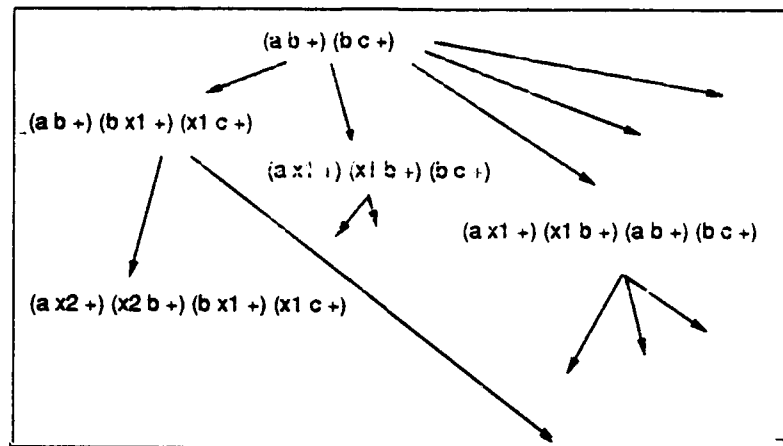


Figure 6-61: Elaboration tree

The system retrieves solutions for three alternative elaborations. This is defined by the variable **elab-alts**. This can be bound to any value as required to generate a specific number of solutions. From a theoretical point of view, issues of interest are the completeness of the search process. The elaboration can also be guided by the use of physical laws. Physical laws indicated certain relationships between parameters and allow only certain kinds of influences. They can be used to instantiate unknowns in the influences and focus the search. Physical laws can also be used to verify the existence of certain influences in a graph for which no matches may be found. If there are physical laws that suggest the feasibility of an influence then it is physically possible to synthesize an artifact that has the concerned influence behavior. This is being implemented as part of the next version of this system.

6.29.2 Retrieval tree

Algorithm RETR_SEARCH creates a *retrieval search* tree rooted at each node of the elaboration tree. The retrieval search tree for the graph $((a\ x1\ +)\ (x1\ b\ +)\ (b\ c\ +))$ is shown in Figure 6-62

The retrieval search tree is destroyed after a search is performed. During the retrieval search process, best-first search is employed. The best node for expansion is chosen based on the case retrieved at the node. A complete topology of both searches is shown in Figure 6-63.

6.30 Useful functions

The retrieval search returns a number of alternative sets of cases that could potentially satisfy a given influence. The number of alternative solutions is defined by the variable **num-alts**. This can be bound to the number of solutions required by the designer. The total number of solutions provided is then the product of the **num-alts** and **elab-alts**. During retrieval, one

[htbp]

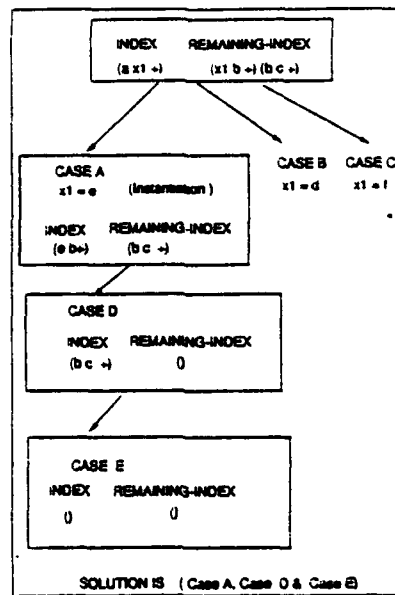


Figure 6-62: Retrieval search tree

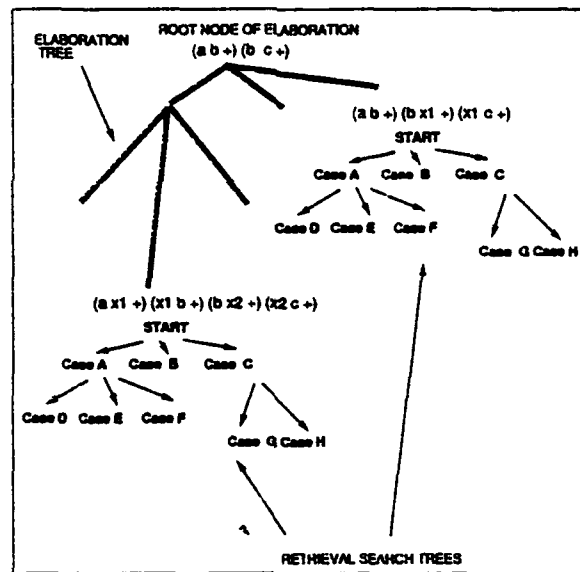


Figure 6-63: Elaboration and retrieval search

might like to impose constraints on the search. These constraints can be of three specific types:

- **Case constraints:** Constraints indicate if a particular case (component) must be part of the solution or not. One might not want a case that has a certain side-effect or a device with a particular geometry.
- **Influence constraints:** Constraints might dictate the inclusion or absence of certain types of influences or relations between parameters.
- **Parameter constraints:** Constraints might dictate the inclusion or absence of certain types of parameters in the influences.

Chapter 7

The Lisp-C-ESQL interface to a Informix Relational Database & CADET Case input and storage mechanisms

7.31 Introduction

CADET is a design tool using the case-based approach. It retrieves relevant cases and their attributes during design based on a number of indices. Case input and retrieval are primary issues in a case-based system. Cases are a high level abstraction of data catering to wide variety of lower level representation mechanisms. The relational database is one of the many data structures that can be used for storing data permanently and also it can handle large data sets. Commercial relational databases are well developed and standardised. Their use guarantees portability of cases and also a uniform language for their input and use. In CADET, cases are stored in a relational database (INFORMIX) as a series of tables. Indices are defined based on the various columns of the tables. The design logic has been implemented in LISP and an efficient interface is required to manipulate and manage data in the relational database from the LISP environment. This document presents a library of functions in LISP that have been implemented using Embedded-SQL (ESQL), C and LISP. Also a case input module has been implemented so as to enable the user to input cases into the database.

The documentation is organised as follows:

- The first section describes some of the essential details regarding the implementation.
- The second section describes all the functions that are available and examples of their usage. Also a brief description of useful SQL commands is provided.
- The third section caters specifically to Case input in CADET.
- Functions to be used for the same are outlined. Also a session with the system is illustrated.

- The final section outlines some of the further additions to be made to the library of functions and highlights the utility of this library.

7.31.1 Implementation issues

Relation databases provide a language called Embedded SQL for accessing databases from programs written in languages like C, ADA and PASCAL. Embedded SQL [informix] provides all the essential functionalities of SQL and a set of mechanisms to handle data IO. Commercial implementations of LISP provide foreign function interfaces to access code written in other languages especially PASCAL, FORTRAN and C. These foreign function interfaces are different from one implementation of LISP to another and minor modifications might be necessary to port from one to the other version. The implementation described herein uses the Allegro Common LISP foreign function interface [allegro], Informix ESQL and ANSI C. Conversion from one data type to another (from the database to C and then to LISP or vice-versa) is a critical issue and varies among the the different versions. While using the library developed these issues will have to be taken care of.

In the set of functions implemented here, the arguments are mainly strings as will be clear in the next section. Parts of a regular SQL statement have been parametrized so that a variety of data manipulation calls can be handled. The code has been tested to some extent and no error handling and recovery mechanisms have been provided for yet. If the function arguments are passed properly, the functions should execute in the expected manner. Otherwise there is an error in function calls.

7.32 Description of LISP-SQL functions

- (CREATEDB "dbname")

•

```
> (create_db ``cadet``)
```

- The above statement creates a database called cadet in a directory defined in the environmental variable DBPATH.

- (CREATETABLE "dbname" "tablename" "("colname" "datatype")")

•

```
> (create_table ``cadet`` ``test1`` ``( "devicename"
```

- The above statement creates a table test1 consisting of two columns devicename and number of the data types defined above.

- (DESTROYDB "dbname")

```
> (destroy_db ``cadet``)
```

- The above statement delates a database called cadet in a directory defined in the

environmental variable DBPATH.

- (DESTROYTABLE "dbname" "tablename")
- > (destroy_table "cadet" "test1")
- The above statement destroys table test1.
- (ADDCOLUMN "dbname" "tablename" "new-column" "new data type")
- > (add_column "cadet" "test1" "device_type" "c
- The above statement adds a new column called devicetype to table test1.
- (DELETECOLUMN "dbname" "tablename" "column")
- > (delete_column "cadet" "test1" "device_type")
- The above statement deletes column called devicetype in table test1.
- (MODIFYCOLUMN "dbname" "tablename" "columnn" "new data type")
- > (modify_column "cadet" "test1" "device_type"
- The above statement changes datatype of column called devicetype in table test1 to integer from what ever it was before.
- (DELETEROWS "dbname" "tablename" "condition")
- > (delete_rows "cadet" "test1" "device_name = "
- Here we delete all rows in table test1 where devicename is TAP . Other conditions as defined in SQL also can be used.
- (INSERTROWS "dbname" "tablename" "(val1,val2)")
- > (insert_rows "cadet" "test1" "("TAP ",12)")
- Here we insert a row in table test1 where devicename is TAP and the number is 12. The list of value should contain values for all columns.
- (SELECTROWS *output-buffer-index* "dbname" "tablename" "conditions")
- > (read-from-string (select_rows *output-buffer-index
device_name = "TAP"))
- A list of unique rows where the devicename is "TAP" is returned.

7.32.1 Error messages

All the above functions except selectrows return a 1 to indicate success and -1 for a failed operation. The function selectrows returns a list of rows satisfying all the conditions or nil as the case may be. Handling of different data types and their conversion schemes have not been debugged completely. They shall be fixed in forthcoming revisions. Care has to be taken about escaping the quotes inside a string. The format statement can be used to create such strings.

7.32.2 Usage of the functions

This interface can be used in a number of ways. It can be used for data retrieval and storage during search (like storing a large set of nodes). The data base can also be used as an additional datastructure for regular programs though question regarding efficiency and speed arise. The data can also be deleted as and when required.

7.33 Cadet case input

A case input module has been implemented for creating cases and loading them into the databases. The cases are stored in a text file and then loaded into the databases. Some essential functions are as follows:

- (INPUTCASES { @em flag}) ; { @em flag} takes values 0 or 1. zero meaning the user wants to input cases and 1 to proceed with the program. (INPUTCASES-FROMFILE "filename" "dbname") inputs cases that have been stored in a file into a database.

The home directory /usr0/madhu has been divided into a number of subdirectories. Please peruse the README file at the top-level. Users work in the Workspace directory. A copy of the text file of cases is stored in ~/textdb. After your session delete the file in the Workspace directory. Take care to have { @em distinct} filenames. Files will be { @em overwritten} if the same names are used. The next section describes an example session.

7.34 An example session

The session transcript is described after logging on to the account. Comments are included in the transcript after a *NOTE* .

```
(CADET.madhu)___> cd Workspace
(CADET.Workspace)___>cl
Allegro CL 3.1.12.2 [DECstation] (11/19/90)
Copyright (C) 1985-1990, Franz Inc., Berkeley, CA, USA
*_CADET_*>(input_cases 0)                                *NOTE* : Note fla

-- CASE INPUT PHASE --
-- CASE TEXT FILE CREATION --
-- CASES WILL BE LOADED INTO DATABASE --

"Do you want to input a case ? Say (y/n) "y

Device name is ? - LIGHT_BULB

Components of the device are - List them with spaces - LIGHT_BULB

Component is LIGHT_BULB
Input number of Edges/Influences ? - 1

Influences are : (left_inf right_inf sign) - ELEC LIGHT +
```

Device type is ? - ELECTRICAL

Component properties input

Component is LIGHT_BULB

Cost of the device is - ? 100

Weight of the device is - ? 12

Material of the device is - ? GLASS

Qual_cost of the device on (1 .. 100) is - ? 1

Qual_weight of the device on (1 .. 100) is - ? 1

Availability of the device on (1 .. 100) is - ? 1

Demand of the device on (1 .. 100) is - ? 1

Reliability of the device on (1 .. 100) is - ? 11

Adapatablity of the device on (1 .. 100) is - ? 1

Performance of the device on (1 .. 100) is - ? 1

"Do you want to input a case ? Say (y/n) "n

--Output file name is ? - data1

NOTE Filename is

*_CADET_*>(input_cases_from_file "data1" "cadet")

NOTE Load cases
into cadet

*_CADET_*> (read-from-string (select_rows *output-buffer-index*
"cadet" "influences" "where device_name = "LIGHT_BULB" "

((LIGHT_BULB LIGHT_BULB ELEC LIGHT +))

NOTE A check if t
have been lo

A suggestion is while experimenting with the input module, create temporary databases or create multiple copies of cadet in the /usr0/madhu/database directory and use them to test the functions.

Your suggestions for modifications are welcome.

Bibliography

- [1] Michael F. Ashby.
On the engineering properties of materials.
Acta metallurgica 37(5):1273--1293, May, 1989.
- [2] Ashby, M.F.
Materials selection oin mechanical design.
Materials Science and Technology 5:517-525, 1989.
- [3] Bagci, C.
Degrees of Freedom of Motion in Mechanisms.
ASME Journal of Engineering for Industry 95:140-48, 1971.
- [4] Bagci, C.
Determining General and Overclosing Constraints in Mechanism Mobility Using
Structural Finite Element Joint Freedoms.
ASME Journal of Mechanical Design 114:376-83, 1992.
- [5] Carbonell, J. G.
Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise
Acquisition.
In Michalski, R. S., J. G. Carbonell, T. M. Mitchell (editor), *Machine Learning: An
Artificial Intelligence Approach Vol 2*. Morgan Kaufman, 1986.
- [6] DeKleer, J., J. Brown .
A Qualitative Physics Based on Confluences.
Artificial Intelligence 24:7-83, 1984.
- [7] Didier Dubois and Henri Prade.
Possibility Theory: An Approach to Computerized Processing of Uncertainty.
Plenum Press, New York, 1988.
- [8] Dunker, K.
On Problem-Solving.
Psychological Monographs 58(270), 1945.
- [9] Finger, S. and J.R. Dixon.
A Review of Research in Mechanical Engineering Design. Part I: Descriptive,
Prescriptive, and Computer-based Models of Design Processes.
Research in Engineering Design 1(1):51-67, 1989.
- [10] Forbus, K.
Qualitative Process Theory.
Artificial Intelligence 24:85-168, 1984.
- [11] Gero, J.S.
Prototypes: a new schema for knowledge-based design.
Technical Report, Architectural Computing Unit, Department of Architectural Science,
Working Paper, 1987.

- [12] Goel, A.K.
Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving.
PhD thesis, The Ohio State University, 1989.
- [13] Goel A., B. Chandrasekaran.
Integrating Model-based Reasoning and Case-Based Reasoning for Design Problem Solving.
In *Proceedings of the AAAI Design Workshop* . Expected in 1989.
- [14] Goel, A.K., J.L. Kolodner, M. Pearce, R. Billington, C. Zimring.
Towards a Case-Based Tool for Aiding Conceptual Design Problem Solving.
In *Proceedings of the 1991 DARPA workshop on Case Based Reasoning*, pages 109-120. 1991.
- [15] Grübler, M.
Getriebelehre. Eine Theorie des Zwanglaufes und der ebenen Mechanismen.
Springer, Berlin, 1921.
- [16] Gregory, S.A.
The Boundaries and Internals of Expert Systems in Engineering Design.
In *Proceedings of the Second IFIP Workshop on Intelligent CAD*. 1988.
- [17] Hammond, K.J.
CHEF: A model of case-based planning.
In *Proceedings of AAAI-86*, pages 267-271. Philadelphia, PA, 1986.
- [18] Hammond, K.J.
CHEF: A Model of Case-based Planning.
In *Proceedings of AAAI-86*, pages 267-271. 1986.
- [19] Hicks, T.G.
Machine Design Calculations Reference Guide.
McGraw-Hill, 1987.
- [20] Hinrichs, T.R., and Kolodner, J.L.
The Roles of Adaptation in Case-Based Design.
In *Proceedings of the Ninth National Conference on Artificial Intelligence*. AAAI, 1991.
- [21] Hix, C.F. Jr., R.P. Alley.
Physical Laws and Effects.
John Wiley and Sons, 1958.
- [22] Howard, C., J. Wang, F. Daube, T. Rafiq.
Applying Design-Dependent Knowledge in Structural Engineering Design.
AI EDAM 3(2):111-123, 1989.
- [23] Huhns M.H., R.D. Acosta.
Argo: An Analogical Reasoning System for Solving Design Problems.
Technical Report AI/CAD-092-87, Microelectronic and Computer Technology Corporation, March, 1987.
- [24] Iwasaki, Y., H.A. Simon.
Causality in Device Behavior.
Artificial Intelligence 29:3-32, 1986.

- [25] Karnopp, D.C. and R.C. Rosenberg.
Systems Dynamics: A Unified Approach.
Wiley, New York, 1975.
- [26] Kolodner, J.L.
Retrieval and organizational strategies in conceptual memory: A computer model.
PhD thesis, Yale University, 1980.
- [27] Kolodner, J. L., Simpson, R. L. Jr., Sycara-Cyransky, K.
A Process Model of Case-Based Reasoning in Problem-Solving.
In *Proceedings IJCAI-9*. Los Angeles, CA, August, 1985.
- [28] Kolodner, J.L.
Extending Problem Solver Capabilities through Case-Based Inference.
In *Proceedings of the 4th Annual In'l Machine Learning Workshop*. Irvine, CA., 1987.
- [29] Kolodner, J.L., Simpson, R.L., and Sycara, K.
A Process Model of Case-Based Reasoning in Problem Solving.
In *Proceedings of IJCAI-85*, pages 284-290. Los Angeles, CA, 1985.
- [30] Kolodner, J.L., Simpson, R.L., and Sycara-Cyranski, K.
A Process Model of Case-Based Reasoning in Problem Solving.
In *Proceedings of the Ninth Joint International Conference on Artificial Intelligence (IJCAI-85)*, pages 284-290. Los Angeles, CA, 1985.
- [31] Kota, S.
A Qualitative Matrix Representation Scheme for the Conceptual Design of Mechanisms.
In *Proc. of the ASME Design Automation Conference (21st Biannual ASME Mechanisms Conference)*. 1990.
- [32] Koton, P.
Using Experience in Learning and Problem Solving.
PhD thesis, M.I.T., 1988.
- [33] Kuipers, B.
Commonsense Reasoning about Causality.
Artificial Intelligence 24, 1984.
- [34] Kuipers, B.J.
Qualitative Simulation.
Artificial Intelligence 29:289-338, 1986.
- [35] Maher, M.L., F. Zhao.
Using Experience to plan the synthesis of new designs.
In Gero, J.S. (editor), *Expert Systems in Computer-Aided Design*. North-Holland, 1987.
- [36] Maier, N.R.F.
Reasoning in humans: II. The solution of a problem and its appearance in consciousness.
Journal of Comparative Psychology 12():181-194, 1931.
- [37] Malysheff, A.P.
Analysis and Synthesis of Mechanisms with the Viewpoint of Their Structures.
Izvestiya Tomskogo of Technological Institute, 1923.

- [38] Michelena, N. and A. Agogino.
Monotonic Influence Diagrams: Foundations and Application to Optimal Design.
1993.
To appear in *Engineering Optimization*.
- [39] Michelena, N. and A. Agogino.
Monotonic Influence Diagrams: Extension to Stochastic Programming and Application to
Probabilistic Design.
1993.
To appear in *Engineering Optimization*.
- [40] Mostow, J.
Toward Better Models Of The Design Process.
The AI Magazine :44-57, Spring, 1985.
- [41] Mostow, J.
Toward Better Models Of The Design Process.
The AI Magazine , Spring, 1985.
- [42] Mostow, J., M. Barley.
Automated Reuse of Design Plans.
In *Proceedings of the International Conference on Engineering Design* . February, 1987.
- [43] Navin chandra, D.
Case-Based Reasoning in CYCLOPS, a Design Problem Solver.
In Kolodner, J. (editor), *Proceedings of the DARPA Workshop on Case-based Reasoning*,
pages 286-301. Morgan Kaufman, 1988.
- [44] Navin Chandra, D.
Exploration and Innovation in Design: Towards a Computational Model.
Springer-Verlag, 1991.
- [45] Navin chandra D., D. Sriram, S.T. Kedar-Cabelli.
On the Role of Analogy in Engineering Design: An Overview.
In D. Sriram, B. Adey (editor), *AI in Engineering, Proceedings of the 2nd Intl.
Conference, Boston*. Computational Mechanics Publishing, U.K., 1987.
- [46] Navin Chandra, D., K. Sycara, S. Narasimhan.
A Transformational Approach to Case Based Synthesis.
AI EDAM 5(1), 1991.
- [47] Navin Chandra D., K.P. Sycara, S. Narasimhan.
Behavioral Synthesis in CADET, A Case-Based Design Tool.
In *Proceedings of the Seventh Conference on Artificial Intelligence Applications*. IEEE,
Miami, Florida, Feb, 1991 .
- [48] Pahl, G., W. Beitz.
Engineering Design.
The Design Council, Springer-Verlag, 1984.
- [49] Paynter, H.M.
Analysis and Design of Engineering Systems.
M.I.T. Press, Cambridge, Mass., 1961.

- [50] Redmond, M.
Distributed Cases for Case-Based Reasoning; Facilitating Use of Multiple Cases.
In *Proceedings of the Eighth National Conference on Artificial Intelligence, AAAI-90*,
pages 304-309. 1990.
- [51] Riesbeck, C.K, R. Schank.
Inside Case-Based Reasoning.
Lawrence Erlbaum Associates, Publisher, 1989.
- [52] Rinderle, J.R.
Measures of Functional Coupling in Design.
PhD thesis, Massachusetts Institute of Technology, 1982.
- [53] Rosenauer, N. and A.H. Willis.
Kinematics of Mechanisms.
Associated General Publications, Sidney, Australia, 1953.
- [54] Rosenberg, R.C. and D.C. Karnopp.
Introduction to Physical Systems Dynamics.
McGraw Hill, New York, 1983.
- [55] Sembugamoorthy V. and B. Chandrasekaran.
Functional Representations of Devices and Compilation of Diagnostic Problem Solving
Systems.
In J. Kolodner, C. Riesbeck (editor), *Experience, Memory, and Reasoning*, pages 47-73.
Earlbaum, Hillsdale N.J., 1986.
- [56] Simoudis, E. , J.S. Miller.
The application of CBR to Help Desk Applications.
In DARPA (editor), *Proceedings of the 1991 Case Based Reasoning Workshop*.
Washington, D.C., 1991.
- [57] Simpson, R.L.
*A Computer Model of Case-Based Reasoning in Problem Solving: An Investigation in the
Domain of Dispute Mediation*.
PhD thesis, School of Information and Computer Science Georgia Institute of
Technology, 1985.
- [58] Steinberg, L.I.
Design as Refinement Plus Constraint Propagation: The VEXED Experience.
In *Proceedings of the sixth national conference on Artificial Intelligence*, pages 830-835.
1987.
- [59] Suh, N.P.
The Principles of Design.
Oxford University Press, Oxford, UK, 1988.
- [60] Suh, N.P., A.C. Bell, D.C. Gossard.
On an Axiomatic Approach to Manufacturing and Manufacturing Systems.
Journal of Engineering for Industry , May, 1978.

- [61] Sycara, K.
Resolving Adversarial Conflicts: An Approach Integrating Case-Based and Analytic Methods.
PhD thesis, School of Information and Computer Science Georgia Institute of Technology, 1987.
- [62] Sycara, K.
Finding creative solutions in adversarial impasses.
In Proceedings of the Ninth Annual Conference of the Cognitive Science Society. Seattle, WA, 1987.
- [63] Sycara, K.
Patching Up Old Plans.
In Proceedings of the Tenth Annual Conference of the Cognitive Science Society. Montreal, Canada, 1988.
- [64] Sycara, K. and Navinchandra.
Index Generation and Transformation for Case retrieval.
In Proceedings of the 1989 Case-Based Reasoning Workshop. Pensacola, Fla., 1989.
- [65] Sycara, K., D. Navin Chandra.
Integrating Case-Based Reasoning and Qualitative Reasoning in Design.
In J. Gero (editor), AI in Design. Computational Mechanics, U.K., 1989.
- [66] Sycara, K. and Navin Chandra, D.
Representing and Indexing Design Cases.
In Proceedings of the Second International Conference on Industrial and Engineering Applications of AI and Expert Systems. Tullahoma, TN, 1989.
- [67] Sycara, K., D. Navin Chandra.
A Process Model of Case Based Design.
In Proceedings of the Cognitive Science Society Conference. Ann Arbor, Michigan, 1989.
- [68] Sycara, K., D. Navin Chandra.
Index Transformation Techniques for Facilitating Creative Use of Multiple Cases.
In Proceedings of the twelfth International Joint Conference on Artificial Intelligence, (IJCAI-91). 1991.
- [69] Sycara, K.P., D. Navin Chandra.
Influences: A Thematic Abstraction for Creative Use of Multiple Cases.
In DARPA Case Based Reasoning Workshop. 1991.
- [70] Sycara, K., D. Navin Chandra.
Index Transformation Techniques for Facilitating Creative Use of Multiple Cases.
In Proceedings of the twelfth International Joint Conference on Artificial Intelligence. 1991.
- [71] Sycara, K., D. Navin Chandra, R. Guttal, J. Koning, S. Narasimhan.
CADET: A Case-Based Synthesis Tool for Engineering Design.
International Journal of Expert Systems 4(2):157-188, 1992.

- [72] Tong, C.
Knowledge-Based Circuit Design.
PhD thesis, Stanford University, 1986.
- [73] Ullman, D.G., T.A. Dietterich.
Mechanical Design Methodology: Implications on Future Developments of Computer-Aided Design and Knowledge-Based Systems.
Engineering with Computers 2:21-29, 1987.
- [74] Ulrich, K. T. and Seering, W. P.
Function Sharing in Mechanical Design.
In *7th National Conference on Artificial Intelligence*. AAAI-88, Minneapolis, MN, August 21-26, 1988.
- [75] Ulrich, K. and W. Seering.
Conceptual Design as Novel Combinations of Existing Device Features.
In *Proceedings of the Advances in Design Automation Conference*. American Society of Mechanical Engineers, Boston, Mass., 1987.
- [76] Ulrich, K. and W. Seering.
Synthesis of Schematic Descriptions in Mechanical Design.
Research in Engineering Design 1(1), 1989.
- [77] Wellman, M.
Fundamental Concepts of Qualitative Probabilistic Networks.
Artificial Intelligence 44(3):257-303, 1990.
- [78] Williams, B.
Invention from First Principles via Topologies of Interaction.
PhD thesis, Massachusetts Institute of Technology, 1989.
- [79] Williams, B.
Interaction-Based Invention: Designing Novel Devices from First Principles.
In *Proceedings of AAAI-90*, pages 349-356. Boston, MA, 1990.